UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Graduado en Ingeniería informática

Clasificación de noticias mediante técnicas de procesamiento del lenguaje natural basadas en aprendizaje profundo

News classification using natural language processing techniques based on deep learning

Realizado por
Pedro Doblas Martín

Tutorizado por
Ezequiel López Rubio

Cotutorizado por
Jorge García González

Departamento
Lenguajes y Ciencias de la Computación

MÁLAGA, junio de 2021

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADUADO EN INGENIERÍA INFORMÁTICA

Clasificación de noticias mediante técnicas de procesamientodel lenguaje natural basadas en aprendizaje profundo

News classification using natural language processing techniques based on deep learning

Realizado por
Pedro Doblas Martín

Tutorizado por
Ezequiel López Rubio

Cotutorizado por
Jorge García González

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO DE 2021

Fecha defensa: julio de 2021

Abstract

The rise of digital journalism and the integration of modern information technologies in web pages have been affecting journalism as a whole in recent years. The advantages of using digital tools to spread news are undeniable, but they also create new problems not present in traditional media. One of these problems is the difficulty of sorting and filtering the different articles, since the amount of news grows each day, as well as the interactions received by these news.

In this dissertation line, we explore the possibility of developing a classifier based on machine learning techniques, with the objective of creating a solution to this problem, carrying out tests with the training parameters and implementing modifications until we reach a final model, capable of taking news headlines and returning an associated category.

The results, although not totally positive, prove that the use of this technology is valid for this purpose. We obtain an approximate precision of 43 %, but we can observe in the results that the errors obtained are, for the most part, due to both imbalances in the data and the semantic similarities between categories.

Keywords: recurrent neural networks, artificial intelligence, machine learning, deep learning, tensorflow, keras, news, digital journalism

Resumen

Uno de los sectores más afectados por la evolución tecnológica de los últimos años es el periodístico, debido tanto al auge del periodismo digital como a la personalización de contenidos mediante el estudio de perfiles de usuario. Las ventajas de utilizar medios digitales para comunicar noticias son innegables, pero también crean nuevos problemas no presentes en los medios de comunicación tradicionales. Uno de esos problemas es la dificultad que supone ordenar y filtrar los distintos artículos, dado que la cantidad de noticias disponibles aumenta día a día, así como las interacciones que reciben.

En este trabajo se ha explorado el desarrollo de un clasificador basado en técnicas de aprendizaje automático, con el objetivo de facilitar la solución de este problema, realizando pruebas con los parámetros de entrenamiento e introduciendo modificaciones hasta llegar a un modelo final, capaz de tomar titulares de noticias y devolver una categoría asociada.

Los resultados, aunque no totalmente positivos, demuestran que el uso de esta tecnología es apropiado para este propósito. Obtenemos una precisión aproximada del 43 %, pero en los resultados podemos observar que los errores obtenidos se deben, en su mayoría, tanto a desequilibrios en los datos como a las similitudes semánticas entre las categorías de las noticias.

Palabras clave: redes neuronales, inteligencia artificial, aprendizaje computacional, tensorflow, periodismo digital

Índice

1

Introducción

Uno de los sectores más afectados por el progreso de las tecnologías de la información ha sido el periodístico. Según la Oficina de Justificación de la Difusión (OJD), el número de medios oficiales de noticias *online* publicados en España ha aumentado en más de un 450 % en los últimos cinco años, así como el volumen de noticias y visitas (Figura 1)

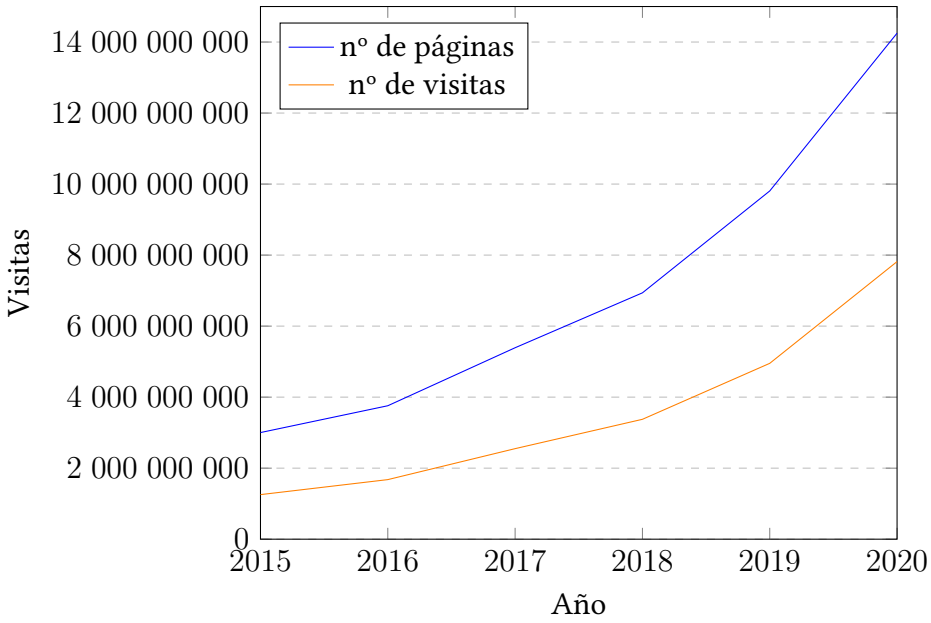


Figura 1: Número de artículos y visitas en medios de noticias egistrados en OJDInteractiva [1]

En los últimos años, el enorme incremento en el uso de internet ha permitido el auge del periodismo multimedia. Este tipo de actividad periodística combina distintos tipos de lenguajes periodísticos (escrito, televisivo...). Una encuesta realizada en 2020 muestra que los periódicos digitales son uno de los medios más utilizados como fuente de noticias (Figura 2).

Esta aplicación de la inteligencia artificial sigue siendo investigada a día de hoy, por lo que exploraremos en este documento una de las posibles aplicaciones dentro de este campo.

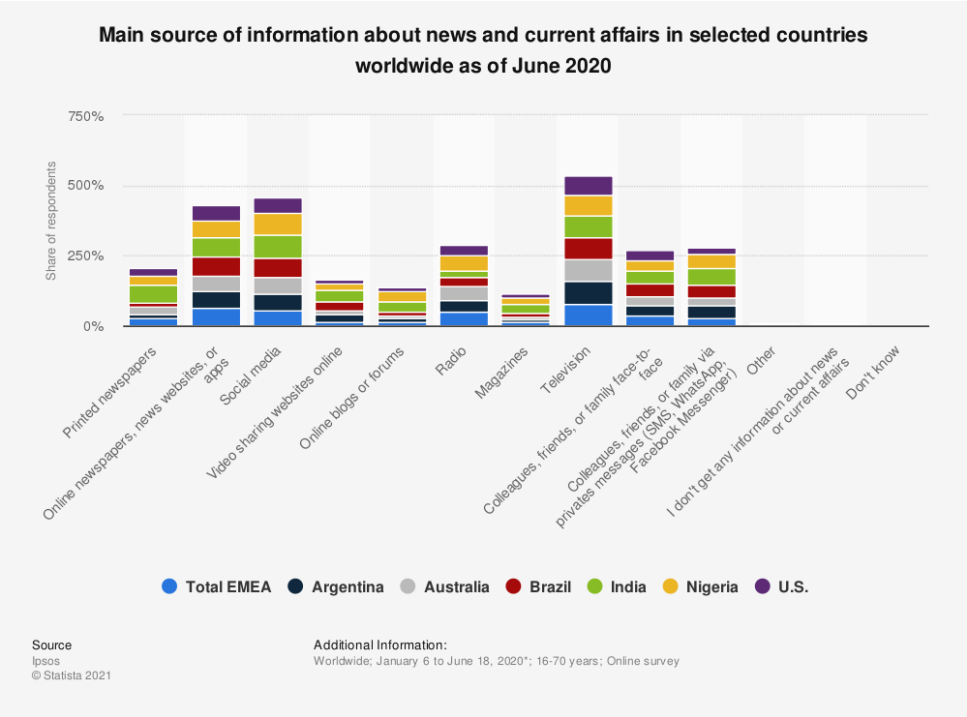


Figura 2: Principal fuente de noticias en múltiples países a nivel mundial (2020)[2]

1.1. Objetivos

La finalidad, tanto de este documento como del código asociado, es explorar una solución a las dificultades asociadas a la organización de noticias en medios periodísticos digitales, mediante el uso de inteligencia artificial y redes neuronales.

A lo largo de los siguientes apartados exponemos el fundamento teórico en el que se basa la realización del trabajo y los distintos modelos creados, así como su estudio y análisis. Finalmente, describiremos las conclusiones obtenidas de estos resultados, así como los modelos que más se aproximan a la solución descrita previamente.

2
Contexto

2.1. Inteligencia artificial

El concepto de inteligencia artificial lleva existiendo desde los inicios de la computación tal y como la conocemos. Una de las primeras denotaciones de este concepto fue creada por Alan Turing en 1950, en sus trabajos de investigación. Turing, uno de los fundadores de la computación y padre de la computadora universal, lanza la pregunta, "¿Pueden las máquinas pensar?"[3]. Esta pregunta, junto a sus otras apreciaciones, comenzarían la definición de uno de los grandes desafíos de la computación.

Actualmente, definimos la inteligencia artificial como el tipo de inteligencia que presentan las computadoras, análoga a la que presentan los seres humanos y los animales. Este tipo de inteligencia está asociada con agentes que pueden realizar determinado tipo de tareas como la toma de decisiones o la percepción del entorno mediante cualquier medio con parámetros que se puedan traducir a una entrada numérica (imágenes, distancias...).

Los inicios del desarrollo de la inteligencia artificial muestran agentes de propósito específico, que previamente debían conocer las reglas del entorno en el que debían operar (por ejemplo, juegos de mesa como el ajedrez). En los últimos años, este paradigma ha cambiado a favorecer el campo conocido como Aprendizaje Computacional (*Machine Learning*), basado en modelos generados independientemente de las reglas, mediante aprendizaje automático.

2.2. Aprendizaje Automático

Podemos definir en el aprendizaje automático varias categorías diferenciadas por los métodos mediante los que se realiza el aprendizaje. Dentro de las principales categorías en las que se dividen los métodos de aprendizaje podemos definir:

- Aprendizaje supervisado: Se toman tanto las entradas como los resultados esperados

para encontrar una función que pueda predecir nuevos datos.

- Aprendizaje no supervisado: El propio sistema obtiene las categorías finales en función de los datos de entrada, encontrando un modelo de densidad asociado.
- Aprendizaje por refuerzo: Se regula el aprendizaje mediante señales de recompensa en función de la proximidad al resultado esperado.

Sin embargo, aunque las tres categorías mencionadas previamente engloban a la mayoría de algoritmos de entrenamiento, los métodos con resultados más relevantes en los últimos años corresponden a la categoría distinguida como Aprendizaje Profundo (*Deep Learning*). Para entenderlos, debemos definir primero el concepto de neurona artificial y red neuronal.

2.3. Redes neuronales

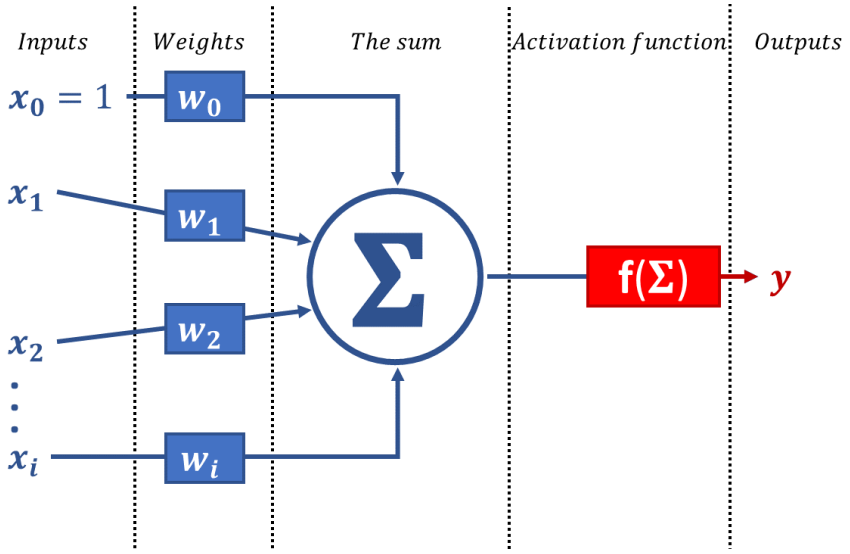


Figura 3: Funcionamiento de una neurona artificial.[4]

Definimos una neurona artificial como una función matemática que toma un conjunto de datos de entrada y los multiplica por unos pesos definidos en la neurona, sumando los resultados y aplicando posteriormente una función de activación que puede ser variable. Esta función de activación es la que determina finalmente la salida de la neurona.

$$y = \emptyset(\sum_{i=1}^{n} w_i x_i)$$

La principal aplicación de las neuronas artificiales a día de hoy se basa en el modelado de redes neuronales, compuestas por múltiples neuronas. Estas redes neuronales se pueden clasificar en varios tipos. Procedemos a exponer las redes neuronales con aplicación en el contexto de este trabajo.

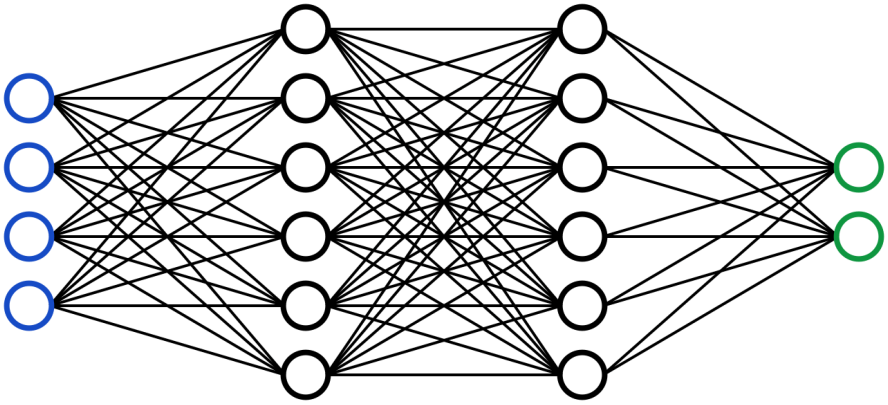


Figura 4: Modelo de red neuronal artificial.[5]

Si tomamos en cuenta este modelo básico de red neuronal (Figura 4), podemos definir además varias capas en su topología. Además de la capa de entrada (en azul) y la capa de salida (en verde), podemos definir el resto de capas como capas ocultas. Estas capas pueden tener varios tipos distintos, definidos por la manera en la que se conectan las neuronas, así como las dimensiones de entrada y salida. La calidad de los resultados obtenidos en una red neuronal está condicionada por varios parámetros como el número de capas, el tipo de estas y la cantidad de neuronas en capa capa. Además, otros parámetros como la función de activación o los tamaños de entrada y salida pueden afectar enormemente al resultado.

2.3.1. Capas Densas

Las capas densas toman todas las entradas de la capa anterior para cada una de las neuronas que contienen. Esto significa que, por defecto, los valores obtenidos dependerán exclusivamente de los pesos y la entrada.

Debido a la gran cantidad de conexiones que contienen, su implementación suele ser computacionalmente costosa. No obstante, este coste sigue siendo menor que el de que otros tipos de capa, debido a la simplicidad en los cálculos a realizar.

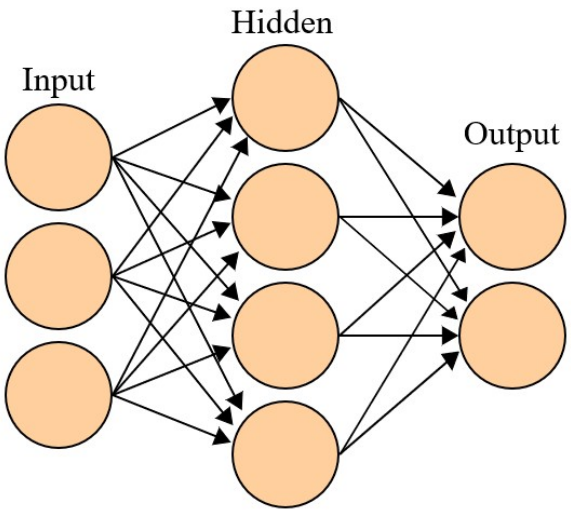


Figura 5: Modelo básico con una capa densa oculta[6]

2.4. Redes neuronales recurrentes

Una red neuronal recurrente (*Recurrent neural networks* o RNN en inglés) es una clase de red neuronal artificial que añade conexiones entre diferentes nodos asignados a distintas capas, de manera que existe una memoria a corto plazo entre las distintas capas, entradas y salida.

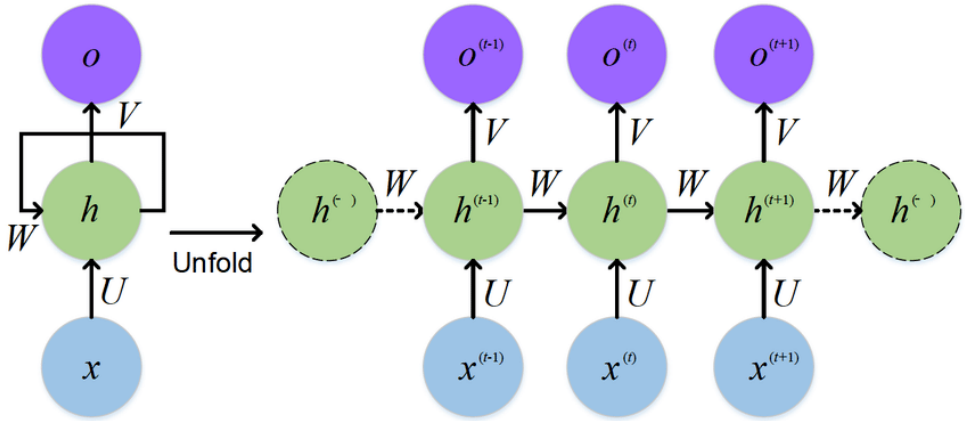


Figura 6: Modelo básico de red neuronal recurrente[7]

Dentro del contexto de este trabajo, las redes neuronales recurrentes son las más prometedoras para procesar lenguaje natural, por lo que serán la opción principal a la hora crear modelos. Existen varios tipos de capas asociados a estas redes neuronales, aunque utilizaremos principalmente las que suelen obtener mejores resultados en el procesamiento del lenguaje

natural.

2.4.1. Capas SimpleRNN

La principal característica definitoria de una red neuronal recurrente es la capacidad de guardar en una memoria información obtenida de los cálculos realizados con información previa presente en la misma tarea.

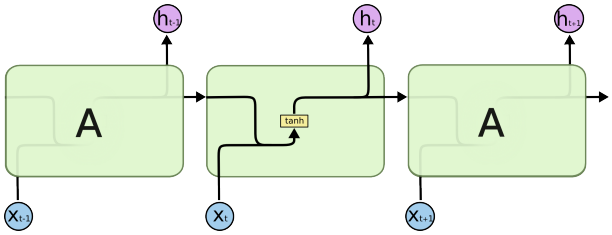


Figura 7: Red neuronal recurrente simple.[8]

Este tipo de capa es el más simple de los que se emplean en las redes neuronales recurrentes, y el método de almacenamiento en memoria suele ser tan sencillo como guardar una única tangente hiperbólica tomada de los datos. El principal problema de emplear un modelo tan sencillo es que no suele ser muy efectivo si necesita guardar memoria a largo plazo, ya que la memoria suele ser sobreescrita en cada paso.

2.4.2. Capas LSTM

Las redes LSTM presentan una solución efectiva al problema presente en las redes neuronales recurrentes simples. Este tipo de capa fue creada por Hochreiter y Schmidhuber en 1997[9].

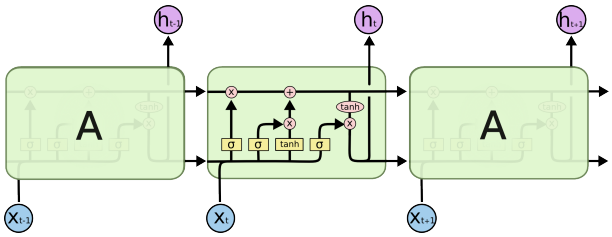


Figura 8: Red neuronal LSTM.[8]

La principal diferencia entre el modelo de red neuronal recurrente simple (Figura 7) y la red neuronal LSTM (Figura 8) es el conjunto de operaciones realizados entre las capas de la

red y la manera en la que se almacenan datos en la memoria. En lugar de almacenar datos en la memoria constantemente, solo se añaden o eliminan datos si se cumplen una serie de condiciones. Así, la red escribe en la memoria exclusivamente cuando hay un cambio relevante, y solo olvida información previa parcialmente, sin eliminarla del todo. En el contexto de este trabajo, utilizaremos la implementación presente en la librería Keras[10].

2.5. Clasificador

Tras operar con las capas expuestas previamente, hay que convertir los datos en un resultado visible. Para ello, la última capa toma las salidas obtenidas y aplica, en nuestro caso, la función **softmax**.

$$\sigma(\overrightarrow{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Esta función toma un vector de K valores y devuelve otro del mismo tamaño, con todos los resultados transformados en números entre 0 y 1, que representan las probabilidades para cada clase.

Esta función puede emplearse en un clasificador únicamente cuando las clases estudiadas son mutuamente exclusivas. En el contexto de este trabajo utilizaremos los resultados de esta función tanto para la fase de entrenamiento como para validación.

2.6. Aprendizaje Profundo

Definimos el concepto de **Aprendizaje Profundo** (*Deep Learning* en inglés) como un conjunto de métodos basados en aprendizaje computacional que incorporan el uso de redes neuronales con grandes cantidades de capas y neuronas. Estos métodos suponen un subconjunto dentro del aprendizaje automático.

La construcción de modelos empleando este tipo de métodos es posible gracias a las grandes mejoras en potencia computacional que se han conseguido en los últimos años.

3

Estado del arte: Procesamiento del lenguaje natural

Definimos el Procesamiento del lenguaje natural (*Natural Language Processing* o NLP en inglés) como la rama de la inteligencia artificial compuesta por el conjunto de técnicas computacionales aplicadas a la comunicación entre máquinas y seres humanos, mediante el uso del lenguaje natural hablado o escrito. Concretamente, está centrado en el procesamiento de datos relacionados con el lenguaje natural, con el objetivo de que la máquina sea capaz de ”entender” los mensajes. Varios de los desafíos más comunes en este campo son el reconocimiento del habla, la comprensión del lenguaje natural escrito y la generación automática de mensajes en lenguaje natural, entre otros.

Hasta finales de los 80, los modelos aplicados para procesar el lenguaje natural estaban basados en conjuntos de reglas introducidas a mano, lo cual hacía que los resultados no fuesen muy prometedores en una gran cantidad de ámbitos. No fue hasta finales de los 80 que se comenzaron a usar modelos basados en aprendizaje automático, que llevaron a resultados mucho mejores. Gracias a las grandes cantidades de datos disponibles a día de hoy, este tipo de modelos se sigue usando en la actualidad con resultados excelentes en múltiples ámbitos. En este caso, utilizamos técnicas de aprendizaje profundo dentro del ámbito periodístico, por lo que es inevitable tener que procesar texto escrito en lenguaje natural. Es por eso que conocer el conjunto de técnicas aplicables resulta de gran relevancia.

Actualmente, la acumulación de datos de redes sociales y otros medios ha cambiado por completo la manera en la que las empresas promocionan sus productos. Surge entonces la necesidad de obtener maneras eficientes de tomar esos datos de manera automática y procesarlos. Las técnicas de procesamiento del lenguaje natural son muy eficientes resolviendo

estos problemas, y es por eso que las empresas que ofrecen servicios relacionados generan una cuota de mercado muy alta [11].

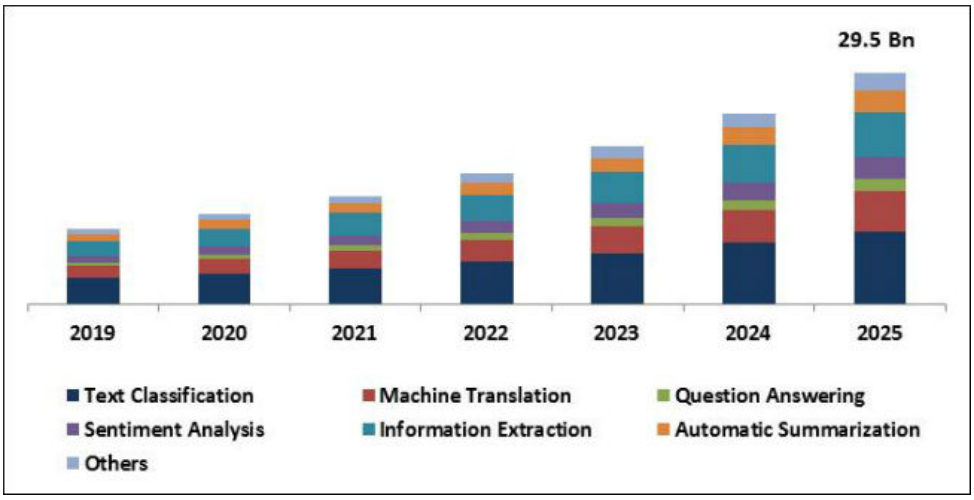


Figura 9: Predicciones sobre la cuota de mercado del procesamiento del lenguaje natural.[11]

También han existido muchos ejemplos de uso de la inteligencia artificial como herramienta de automatización para redacción de documentos periodísticos. Uno de los ejemplos relevantes más reciente sería el software Dreamwriter, capaz de escribir artículos de manera automática y creado por la empresa Tencent en 2018[12].

4

Descripción de los datos empleados

El conjunto de datos o *dataset* empleado contiene 202372 noticias escritas en Huffpost, uno de los periódicos digitales más populares en los Estados Unidos.

Además del titular, el conjunto de datos también contiene la categoría, autores, enlace, descripción y fecha de cada artículo. Las fechas de estos artículos varían entre el año 2012 y el año 2018, y los datos fueron tomados en 2018 por Rishabh Misra, usuario de la plataforma Kaggle de la que tomamos los datos para este trabajo[13].

El método más común para obtener este tipo de información es el *web scraping* o raspado web. Esta técnica utiliza una serie de métodos como el uso de expresiones regulares y reconocimiento automático de información semántica, con el objetivo de extraer información de sitios web.

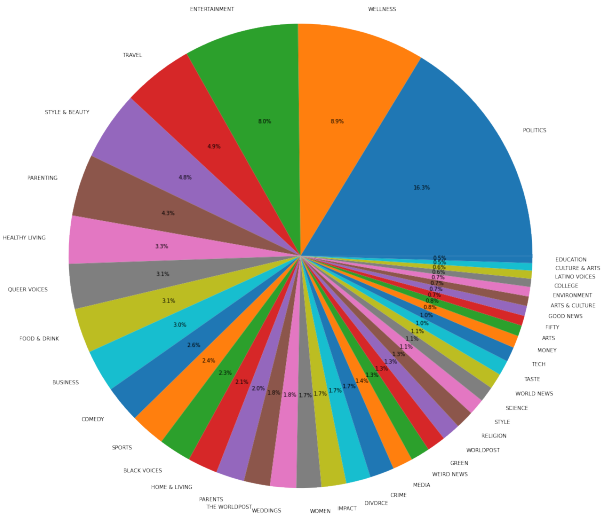


Figura 10: Frecuencia relativa de las categorías en el dataset

Categoría	Número de datos
POLITICS	32739
WELLNESS	17827
ENTERTAINMENT	16058
TRAVEL	9887
STYLE & BEAUTY	9649
PARENTING	8677
HEALTHY LIVING	6694
QUEER VOICES	6314
FOOD & DRINK	6226
BUSINESS	5937
COMEDY	5175
SPORTS	4884
BLACK VOICES	4528
HOME & LIVING	4195
PARENTS	3955
THE WORLDPOST	3664
WEDDINGS	3651
WOMEN	3490
IMPACT	3459
DIVORCE	3426
CRIME	3405

Categoría	Número de datos
MEDIA	2815
WEIRD NEWS	2670
GREEN	2622
WORLDPOST	2579
RELIGION	2556
STYLE	2254
SCIENCE	2178
WORLD NEWS	2177
TASTE	2096
TECH	2082
MONEY	1707
ARTS	1509
FIFTY	1401
GOOD NEWS	1398
ARTS & CULTURE	1339
ENVIRONMENT	1323
COLLEGE	1144
LATINO VOICES	1129
CULTURE & ARTS	1030
EDUCATION	1004

Cuadro 1: Número de datos para cada categoría.

En el contexto de este trabajo, solo tendremos en cuenta el titular y la categoría, ya que corresponden con la entrada y la salida de los modelos que queremos obtener. Dado que las cantidades de artículos para cada categoría son muy distintas, tomaremos un subconjunto de estos con el objetivo de igualarlas. También haremos una serie de modificaciones en la estructura de los datos con intención de mejorar el entrenamiento. Estas modificaciones consistirán en eliminar caracteres no alfanuméricos, hacer que todas las letras de los titulares estén en

minúscula, y eliminar tanto adverbios como preposiciones. Además, para entrenar los modelos dividiremos los datos en conjuntos de entrenamiento y validación, correspondiendo estos a un 70 y un 30 por ciento, respectivamente.

Cabe destacar que tanto las diferencias en la frecuencia de los datos como la similitud semántica entre las categorías son problemas que deberán tenerse en cuenta al procesar los datos. Más adelante detallamos cómo intentamos solucionar esto, así como la calidad de los datos en el contexto de la tarea prevista.

5

Métodos empleados

5.1. Preprocesado de los datos

Uno de los errores que más se han repetido a lo largo de las pruebas realizadas en los modelos corresponden a determinadas categorías que conllevan una semántica igual o similar. Dado que una noticia puede pertenecer a varias categorías, se han realizado unificaciones en estas categorías con el objetivo de obtener una clasificación más precisa.

Las unificaciones realizadas en las categorías han sido:

- ”WORLD NEWS” , ”THE WORLDPOST” y ”WORLDPOST” se han unido en la categoría ”WORLD NEWS”
- ”ARTS & CULTURE”, ”ARTS” y ”CULTURE AND ARTS” se han unido en la categoría ”CULTURE AND ARTS”
- ”FOOD AND DRINK” y ”TASTE” se han unido en la categoría ”FOOD AND DRINK”
- ”STYLE AND BEAUTY” y ”STYLE” se han unido en la categoría ”STYLE AND BEAUTY”
- ”HEALTHY LIVING” y ”WELLNESS” se han unido en ”HEALTHY LIVING”
- ”COLLEGE” y ”EDUCATION” se han unido en ”EDUCATION”
- ”PARENTS” y ”PARENTING” se han unido en ”PARENTING”
- ”GREEN” y ”ENVIRONMENT” se han unido en ”ENVIRONMENT”

Estos cambios han sido realizados dentro de la ejecución del código, por lo que la entrada de este seguirá siendo el dataset original. Además, debido a la gran disparidad entre las cantidades de datos que hay para cada categoría, hemos tomado grupos de mil artículos para

cada categoría. De este modo están más equilibrados, aunque se utilice una menor cantidad de datos.

También hemos eliminado tanto los caracteres no alfanuméricos como los adverbios de los titulares, con el objetivo de que no interfieran en el entrenamiento. Finalmente, hemos eliminado los elementos vacíos y distribuido de manera aleatoria los datos para intentar que no estén agrupados por categorías.

5.2. Tokenización y *Word embeddings*

Teniendo en cuenta que las entradas de una red neuronal deben ser numéricas, necesitamos una manera efectiva de convertir un texto en un conjunto de números. La tokenización consiste en dividir el texto en componentes, que pueden ser palabras, signos de puntuación, etc. En nuestro caso, hemos utilizado la implementación en Keras de un tokenizador de texto.

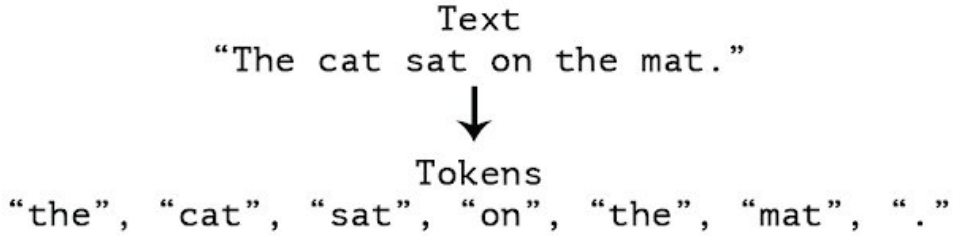


Figura 11: Ejemplo de texto tokenizado.[14]

La tokenización obtenida consiste en un conjunto de números que representan conjuntos de caracteres, así como un diccionario que nos indican la representación exacta de los mismos.

Definimos, por otra parte los *word embeddings* como representaciones de palabras en un texto, teniendo en cuenta los contexto y las asociaciones que tienen. Estas representaciones se muestran mediante vectores que relacionan unas palabras con otras, y se deben tener en cuenta para dar más o menos importancia a determinadas palabras en un contexto.

En nuestro caso, utilizaremos la combinación de ambas. La tokenización se realizará en el preprocesado de los datos, mientras que los *word embeddings* se utilizan en los modelos, entrenándose junto al modelo mediante la implementación en Keras [15].

5.2.1. Palabras fuera de vocabulario

Dado que la tokenización se realiza exclusivamente en el conjunto de datos de entrenamiento, es bastante probable que la red encuentre palabras que no están en el diccionario. Por defecto, el tokenizador devuelve el conjunto de números ignorando estas palabras, pero en el contexto de este trabajo hemos añadido un token universal que se utiliza en lugar de estas palabras. Así, se tiene en cuenta la longitud de los textos incluso si contienen palabras desconocidas para la red.

5.3. Entrenamiento y validación

Tanto entrenar una red como comprobar los resultados que obtenemos de ella necesitamos conjuntos de datos. Si utilizamos el mismo conjunto de datos en ambos casos, los los resultados obtenidos no describirán correctamente la capacidad de la red en el problema para casos fuera del conjunto suministrado.

Para evitar este problema, separamos el conjunto de datos inicial en varias partes. Estas partes representan los conjuntos suministrados para entrenamiento, validación y test, siendo generalmente el conjunto de entrenamiento el mayor de los tres. En nuestro caso, la división realizada hace que el conjunto de datos empleados para entrenamiento y validación representen un 80 %, mientras que el conjunto de test representa un 20 %.

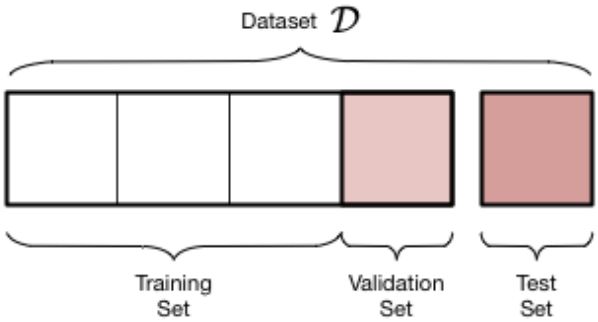


Figura 12: División de los datos.[16]

Una vez separados los datos, podemos entrenar la red utilizando las dos primeras partes, y estudiar los resultados suministrando a la red el conjunto definido para test.

5.4. Optimizador

El optimizador escogido para el desarrollo de estos modelos ha sido el optimizador Adam. Estos optimizadores cuentan con un valor numérico denominado tasa de aprendizaje, que representa la magnitud de los cambios realizados en la red para cada paso. El uso de números mayores acelera el aprendizaje, pero reduce la precisión.

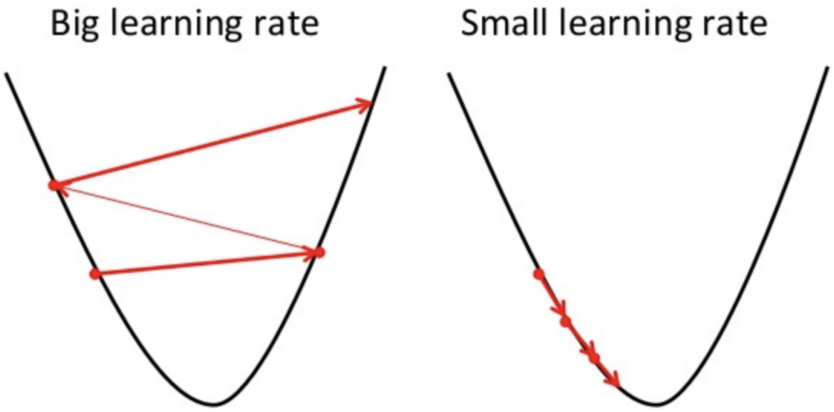


Figura 13: Efectos de la tasa de aprendizaje en el descenso de la gradiente.[17]

En nuestro caso, se ha elegido una tasa de aprendizaje de 1e-3, dado que los resultados han sido muy similares incluso tomando tasas de menor tamaño, lo cual ha llevado a tiempos de entrenamiento muy elevados sin mejoras notables.

5.5. Prevención del sobreajuste

Uno de los principales problemas a evitar en el desarrollo de modelos de red neuronal es el sobreajuste de la red (*overfitting* en inglés). Este problema surge del aprendizaje excesivo sobre el conjunto de entrenamiento, lo que lleva a que la red pierda eficacia sobre datos no estudiados previamente.

Definimos la función de perdida o coste como la operación matemática que cuantifica los errores que ha cometido la red. Esta función se aplica por separado tanto en el entrenamiento como en la validación, y nos sirve para evaluar el entrenamiento de la red a medida que este avanza.

Durante el entrenamiento de un modelo de red neuronal, el valor de la función de pérdida disminuye tanto en los datos de entrenamiento como en los de validación. Llegado cierto mo-

mento, la función de pérdida comienza a obtener valores crecientes en los datos de validación, aunque sigan decreciendo para los datos de entrenamiento. Es aquí cuando consideramos que la red empieza a ajustarse en exceso.

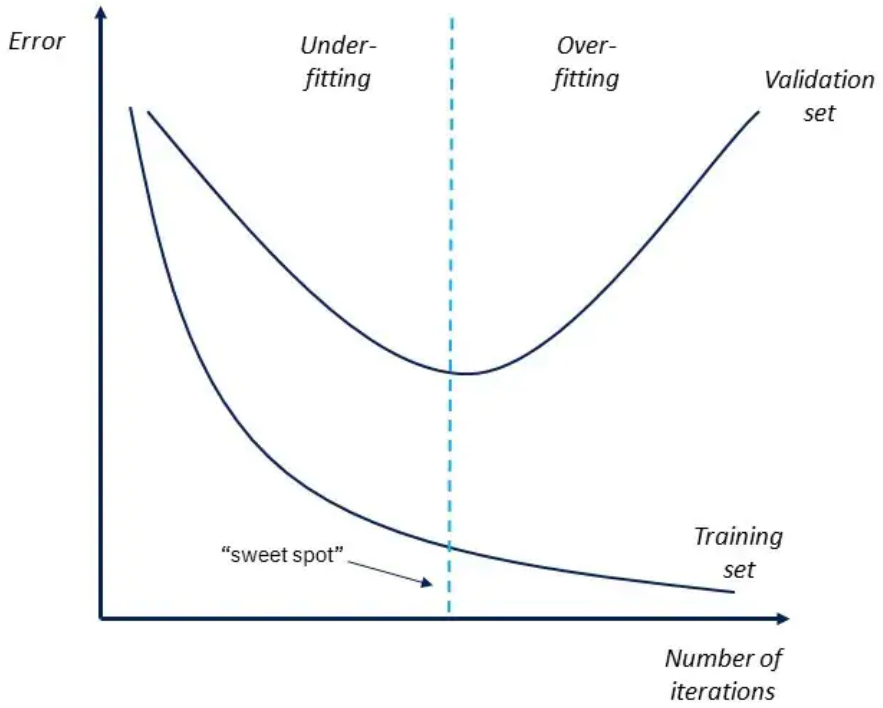


Figura 14: Sobreajuste en la función de pérdida.[18]

Para evitar estos resultados, hemos especificado en el entrenamiento la detención de este en el caso de que el resultado para la función de pérdida en la validación aumente. Así, intentamos aproximarnos en la medida de lo posible al mínimo absoluto de esta función, evitando el sobreajuste de la red.

En el contexto de este trabajo, la función de pérdida que emplearemos será la función *categorical crossentropy*, debido a su eficacia en problemas con múltiples categorías. Esta función intenta medir la diferencia entre las distribuciones de probabilidad esperadas y las obtenidas mediante una función de activación (en nuestro caso softmax), mediante la fórmula:

$$Loss = -\sum_{i=1}^{n} y_i \log \hat{y}_i$$

Donde y_i es la probabilidad de la predicción para la clase i, $\hat{y}_i$ es la probabilidad deseada para la clase i (será 1 sólo cuando sea la clase correcta), y n es el número de clases que tenemos.

5.6. Evaluación del modelo

Una vez entrenados los modelos, necesitamos una manera de medir los resultados obtenidos y tomar decisiones en cuanto a los cambios a realizar. Para las medidas que se calculan en cada categoría, al tener una gran cantidad de estas, utilizaremos la media de los valores obtenidos para cada una.

5.6.1. Medidas aritméticas

Definimos la **precisión** (*precision* en inglés) como el total de verdaderos positivos entre la suma de verdaderos positivos y falsos positivos.

$$Precisión = \frac{VP}{VP + FP}$$

Indica la calidad del resultado obtenido, es decir, la probabilidad estadística de que el resultado obtenido sea correcto.

Calculamos la **exhaustividad** (*recall* en inglés) dividiendo la cantidad de verdaderos positivos entre la suma de verdaderos positivos y falsos negativos.

$$Exhaustividad = \frac{VP}{VP + FN}$$

Esta medida representa la cantidad de positivos que el modelo es capaz de acertar.

La **exactitud** (*accuracy*) mide el porcentaje de casos que el modelo ha acertado en su totalidad.

$$Exactitud = \frac{VP + VN}{VP + VN + FP + FN}$$

Aunque es muy útil para conocer la calidad del modelo de una forma muy general, es muy sensible al desbalance en la cantidad de clases y puede llevar a conclusiones equivocadas.

Utilizamos el **valor F** (*F1 Score* en inglés) para combinar en un mismo valor la precisión y la exhaustividad, asumiendo que damos la misma importancia a ambas. Se calcula mediante

la media armónica de ambas medidas.

$$Valor - F = 2 * \frac{Precisión * Exhaustividad}{Precisión + Exhaustividad}$$

5.6.2. Matriz de confusión

Una matriz de confusión toma el conjunto de predicciones de un modelo y los resultados esperados (reales), mostrando las cantidades de aciertos y errores para cada clase.

	Predicción 0	Predicción 1
Valor real 0	Verdadero negativo	Falso positivo
Valor real 1	Falso negativo	Verdadero positivo

Cuadro 2: Matriz de confusión para clasificación binaria

Los datos obtenidos en la diagonal de la matriz definen los aciertos para cada clase, mientras que los demás elementos definen los errores ha habido y qué clase se ha confundido con otra. Así, podemos destacar con facilidad cuales son las confusiones más comunes del modelo.

5.7. Métodos tradicionales

Además de los modelos basados en aprendizaje automático, se han utilizado otros métodos de clasificación matemáticos tradicionales. Aunque menos eficaces que los métodos de aprendizaje profundo, tienen menor coste computacional.

5.7.1. Random forest

Random forest es un algoritmo de aprendizaje supervisado basado en árboles de decisión. Estos árboles consisten en modelos de predicción expuestos mediante diagramas de construcciones lógicas.

Cada rama del árbol representa una posible decisión, conclusión o reacción, y cada hoja representa el resultado asociado a las decisiones tomadas.

El algoritmo Random forest crea varios árboles de decisión, que toman parámetros del dataset de manera aleatoria. Finalmente, estos árboles se unen en un único árbol, que finalmente se poda. La predicción de este modelo se basa en los resultados de cada uno de los árboles unidos.

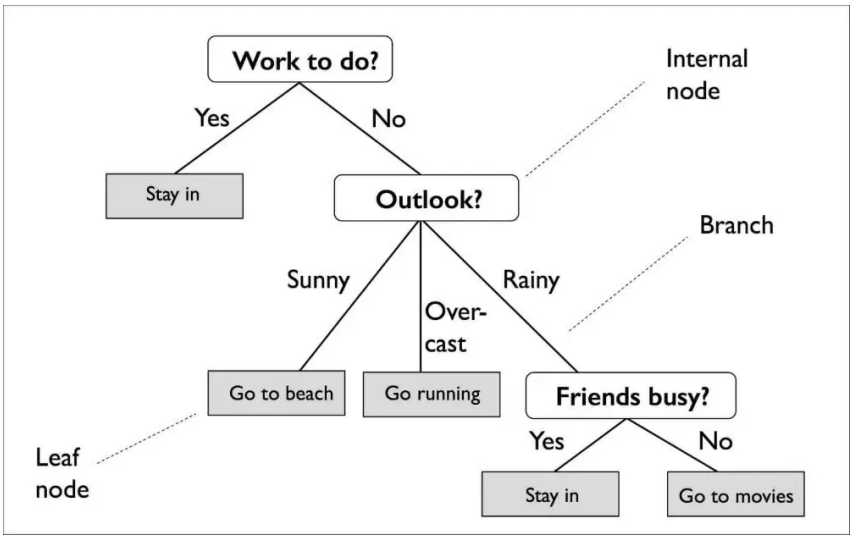


Figura 15: Ejemplo de árbol de decisión.[19]

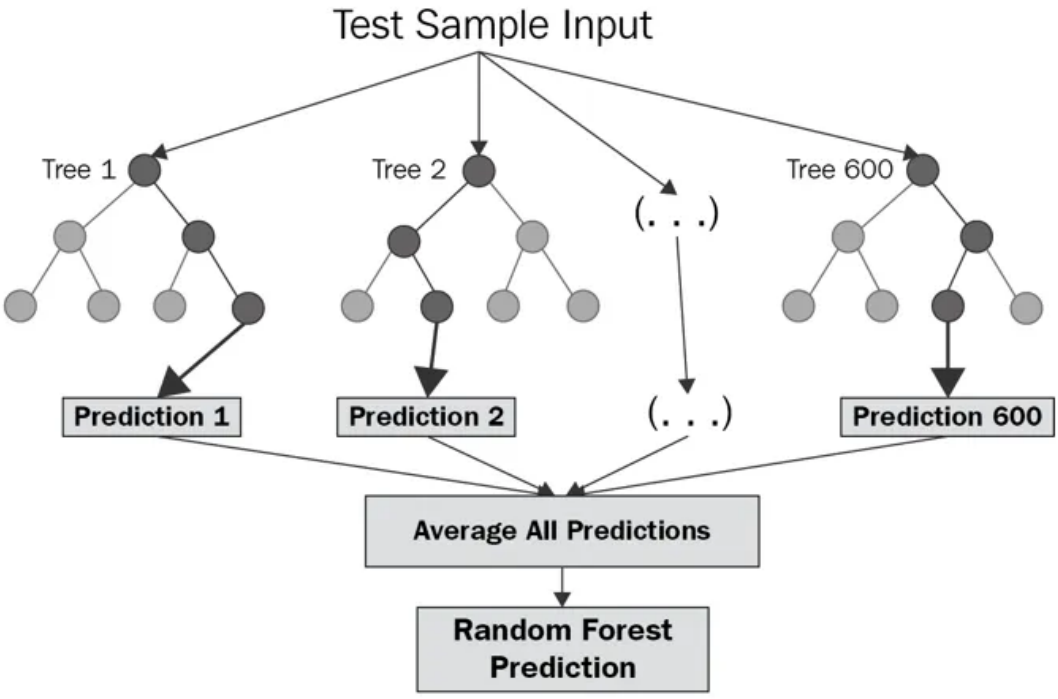


Figura 16: Funcionamiento del algoritmo *Random forest*.[20]

5.7.2. Regresión logística

La regresión logística es un algoritmo de clasificación basado en estadística que permite encontrar relaciones entre dos variables (dependiente e independiente). Concretamente, el algoritmo intenta encontrar una función sigmoide que pueda clasificar los datos de entrena-

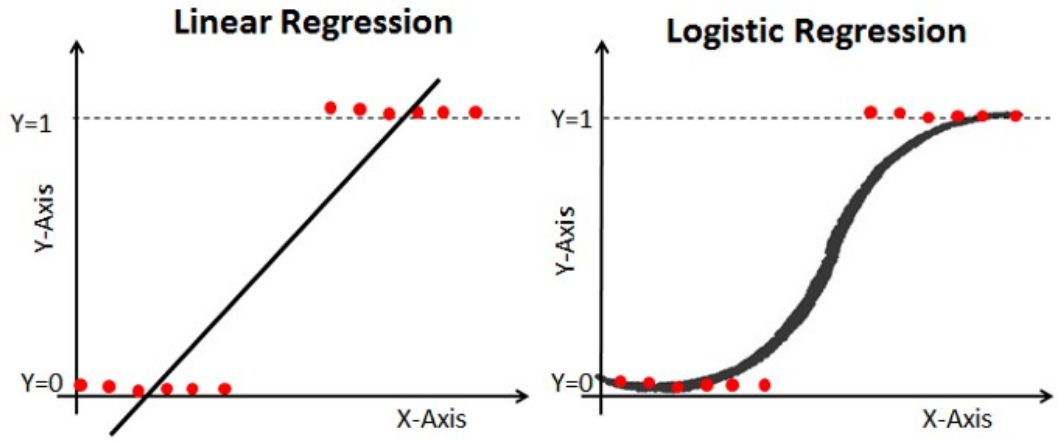


Figura 17: Funciones lineal y sigmoide asociadas a la regresión lineal y logística.[21]

miento. Aunque suele funcionar mejor con modelos con pocas categorías, hemos intentado aplicarlo en este problema para comprobar los resultados obtenidos.

5.7.3. Gaussian Naive Bayes

Definimos los **clasificadores de Bayes** como una familia de clasificadores +estadísticos, cuyo objetivo es encontrar la probabilidad de que un grupo de datos pertenezca a una clase. Si suponemos que hay una independencia fuerte entre las características de los datos estudiadas, nos referimos específicamente a los **clasificadores de Bayes ingenuos**. Ambos están basados en el teorema de Bayes, que describe la probabilidad de un evento en función de las condiciones previas relacionadas con este. El teorema en cuestión tiene la formula siguiente:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Donde A y B son eventos, $P(A)$ es la probabilidad de que el evento A ocurra y $P(B) \neq 0$.

En el caso más concreto del clasificador gaussiano, el algoritmo intenta encontrar una distribución gaussiana para cada clase, dentro de los parámetros suministrados. Así, podemos definir una probabilidad de que un elemento pertenezca a a una clase dependiendo de la entrada.

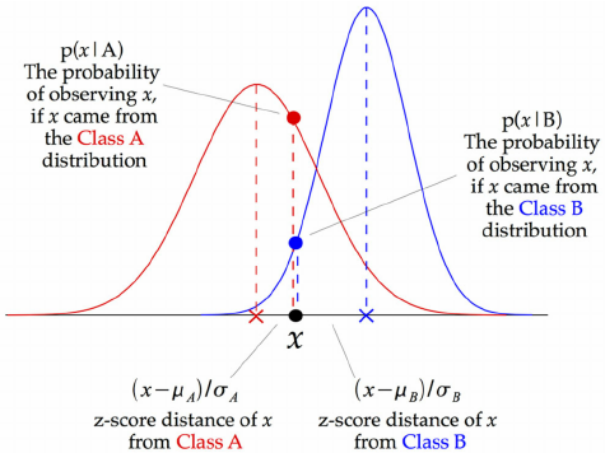


Figura 18: Funcionamiento de un clasificador Gaussian Naive Bayes.[22]

5.7.4. Máquina de vectores de soporte

Una máquina de vectores de soporte (*support vector machine* en inglés) es un modelo de aprendizaje automático que toma un conjunto de datos y crea hiperplanos que separan estos datos según las clases que debamos obtener. El coste que supone crear estos modelos es mucho menor que el empleado para entrenar redes neuronales.

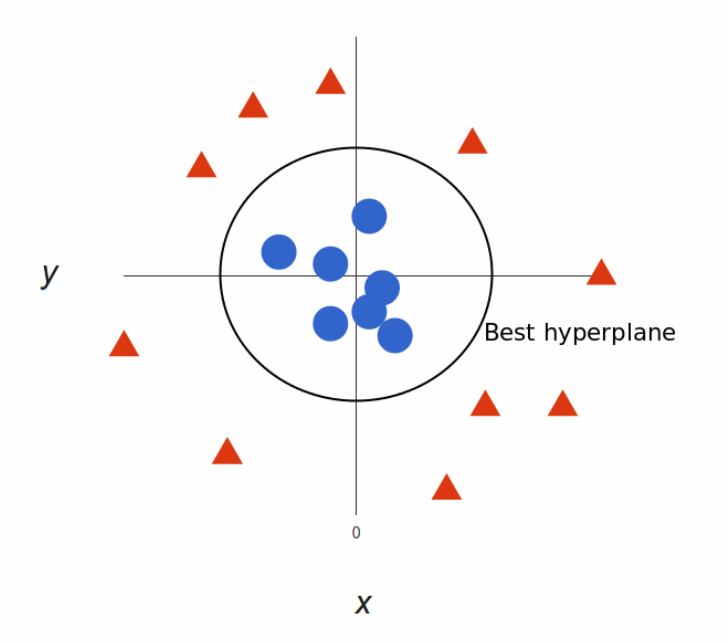


Figura 19: Ejemplo de hiperplano creado para un problema sencillo con dos clases.[23]

5.8. Tecnologías usadas

Para la realización del trabajo se ha utilizado el lenguaje de programación Python, así como una serie de librerías para modelar, probar y guardar los resultados de los métodos explicados previamente:

- Tensorflow y Keras
- Sklearn
- Numpy
- Pandas
- Matplotlib
- Seaborn
- Pickle
- NLTK (*Natural Language Toolkit*)

La mayor parte del desarrollo del trabajo se ha realizado mediante el uso de las librerías **Tensorflow** y **Keras**. La utilidad principal de estas librerías es la de construir y entrenar redes neuronales, empleando como estructuras de información arrays multidimenstionales de datos, referidos como tensores. Estas librerías son de código abierto y se pueden ejecutar en una gran multitud de procesadores y tarjetas gráficas.

Para la lectura y procesado de los datos, utilizamos también las librerías **numpy** y **pandas**, ambas orientadas al procesamiento numérico de matrices y conjuntos de datos. También empleamos la librería **sklearn**, orientada al procesamiento de datos en el entorno del aprendizaje automático, además de ofrecer funciones para probar los métodos tradicionales propuestos.

Las librerías **Matplotlib** y **Seaborn** han sido empleadas para exponer las gráficas y las matrices de confusión empleadas en la evaluación y descripción de los modelos.

Finalmente, la librería **Pickle** se ha utilizado para guardar y cargar la información correspondiente al historial de entrenamiento de los modelos, presente en los archivos incluídos junto al código.

Todo el desarrollo del código se ha realizado en la plataforma Google Colab, que permite utilizar recursos computacionales ofrecidos por Google para crear documentos con el formato abierto ofrecido por Jupyter Notebook. Estos documentos incorporan una combinación de texto y código en el mismo documento, separándolos en distintas celdas.

6

Pruebas realizadas y resultados

Tras comenzar con el modelado de redes neuronales sencillas, podemos utilizar las medidas expuestas en la sección 5.6 para realizar cambios e intentar obtener mejores resultados. En las tablas del apéndice B tenemos la lista completa de todos los modelos empleados y los resultados obtenidos.

Como se ha descrito en la sección 5, para todos los modelos estudiados la función de activación empleada ha sido softmax y la función de pérdida ha sido la función *categorical crossentropy*. Además, tanto los datos de entrada como las medidas estudiadas también han sido las mismas.

El proceso mediante el que se han ido creando estos modelos comienza con un modelo básico con una sola capa densa. A partir de ahí, se han ido realizando cambios en el modelo y documentando los resultados obtenidos, asignando nombres a los modelos en función de la prueba realizada y el orden en el que han sido entrenados.

Para ordenar los modelos, estos han sido agrupados en secciones de la A a la I, ordenadas en orden alfabético y correspondientes a cada prueba realizada. Los nombres de los modelos tienen la letra de la sección a la que pertenecen y el número de orden en el que fueron entrenados. Por ejemplo, el modelo A2 pertenece a la sección A (de la primera prueba realizada) y fue el segundo en ser entrenado dentro de esa prueba.

En todos los modelos, se ha utilizado la misma entrada, tokenización, capa de *embedding* (ver sección 5.2) y capa de salida densa. Además, excepto en los modelos de la sección H la tasa de aprendizaje empleada ha sido 1^{-3}. Para la sección H, explicaremos qué cambios se han hecho en este parámetro y a qué modelos se aplican.

6.1. Redes neuronales simples

6.1.1. Aumento de neuronas en modelos de una sola capa densa (Sección A)

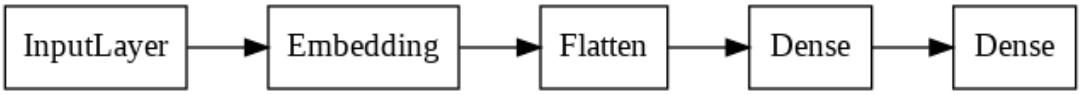


Figura 20: Arquitectura en Keras de los modelos en la sección A

Esta sección se centra en el modelo más básico de red neuronal, con una única capa oculta densa. Los datos obtenidos del *embedding* se pasan a un vector de una dimensión mediante el uso de la capa *flatten*, para usarlo finalmente como entrada de la capa densa oculta.

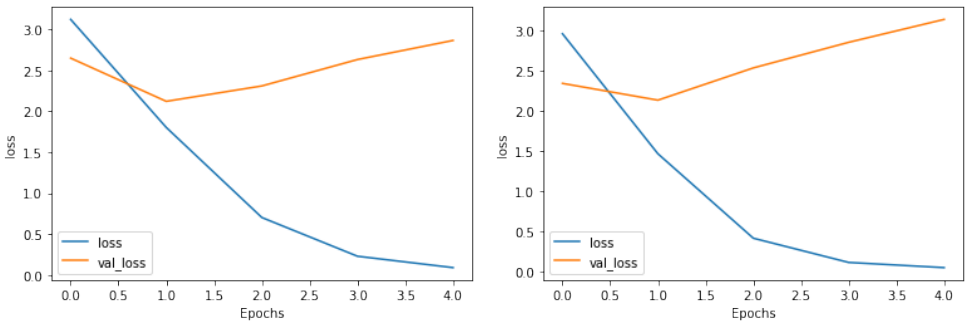
Los modelos A1-A4 pertenecientes a esta sección se diferencian únicamente en el número de neuronas que hay en la capa oculta, ya que queremos encontrar las diferencias de eficacia asociadas con este parámetro. Los números de neuronas probados han sido 128, 256, 512 y 1024.

Modelo	Neuronas en capa densa	Test Accuracy	Test Loss	Precision	Recall	F1-Score
A1	128	42.14 %	2.142	0.47	0.42	0.43
A2	256	43.31 %	2.112	0.46	0.43	0.44
A3	512	42.90 %	2.139	0.46	0.43	0.44
A4	1024	43.14 %	2.168	0.45	0.43	0.43

Cuadro 3: Resultados obtenidos en los modelos de la sección A.

De estos modelos (tabla 3) podemos destacar los resultados de los modelos A2 y A4. El modelo A2 es el que ofrece los mejores resultados en exactitud y valor de la función de pérdida por un pequeño margen. Además, si tenemos en cuenta la proximidad en el resto de medidas para todos los modelos, podemos suponer que han aprendido de manera muy similar.

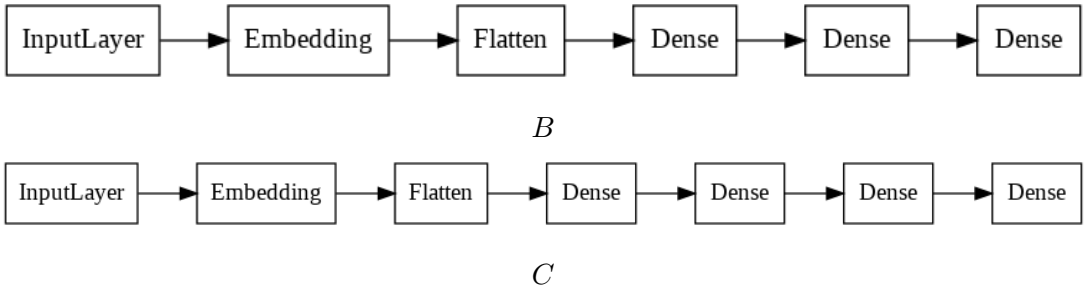
Debido a que el modelo A2 ha encontrado mínimo absoluto de menor tamaño que el resto, concluimos entonces que 256 es el mejor tamaño de capa entre los que hemos probado. En los siguientes apartados utilizaremos el modelo A2 como ejemplo para comparar los resultados obtenidos, hasta que encontremos otro modelo que lo supere.



(a) $A2$ (b) $A4$

Figura 21: Gráficas obtenidas de la función de pérdida en los modelos A2 y A4.

6.1.2. Aumento del número de capas (Secciones B y C)



B

C

Figura 22: Arquitecturas de los modelos en las secciones B y C.

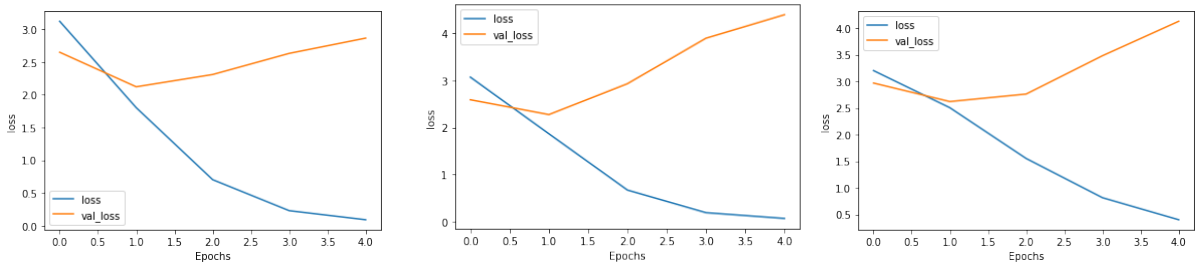
Tras comparar modelos de una sola capa, hemos probado añadiendo más capas densas con el objetivo de obtener una mejora en el aprendizaje, obteniendo como resultado los modelos de las secciones B y C. Los de la sección B tienen dos capas densas ocultas, mientras que los modelos de la sección C tienen tres. Exceptuando el número de capas, los parámetros empleados han sido exactamente los mismos que en la sección anterior.

Aunque generalmente no muy buenos, los mejores resultados obtenidos de esta prueba han sido los del modelo B4 para las redes de dos capas, y los del modelo C3 para redes de tres capas. Teniendo en cuenta la diferencia con los resultados obtenidos respecto al modelo A2, podemos concluir que al aumentar el número de capas el entrenamiento de la red empeora.

Si observamos las gráficas correspondientes a la función de pérdida (figura 23), notamos que al aumentar el número de capas el valor de pérdida para validación decrece mucho menos, encontrando un mínimo mucho peor que el obtenido en el modelo de una sola capa.

Modelo	Nº capas	Tamaño de capa	Test Accuracy	Test Loss	Precision	Recall	F1
B1	2	128	37.64 %	2.344	0.40	0.38	0.38
B2	2	256	37.18 %	2.332	0.37	0.37	0.36
B3	2	512	37.48 %	2.347	0.40	0.38	0.37
B4	2	1024	38.53 %	2.330	0.39	0.39	0.38
C1	3	256	26.86 %	2.629	0.31	0.27	0.25
C2	3	512	24.72 %	2.676	0.26	0.25	0.23
C3	3	1024	24.07 %	2.708	0.27	0.24	0.22

Cuadro 4: Resultados obtenidos en los modelos de las secciones B y C



(a) $A2$ (b) $B4$ (c) $C3$

Figura 23: Gráficas obtenidas de la función de pérdida en los modelos A2, B4 y C3.

A la vista de los resultados, concluimos que no hay necesidad de hacer más pruebas con los tamaños de estas capas. En su lugar intentaremos cambiar la arquitectura utilizando redes neuronales recurrentes.

6.2. Redes neuronales recurrentes (Sección D)

Una vez estudiados los modelos de red neuronal con capas densas, conviene hacer otro tipo de pruebas con otras redes neuronales e intentar mejorar los resultados previos. La siguiente prueba realizada ha consistido en el uso de redes neuronales recurrentes, en principio empleando los modelos más sencillos de este tipo, sin ninguna mejora en la memoria de este tipo de redes.

Al contrario que en los modelos anteriores, para las redes neuronales recurrentes no hace

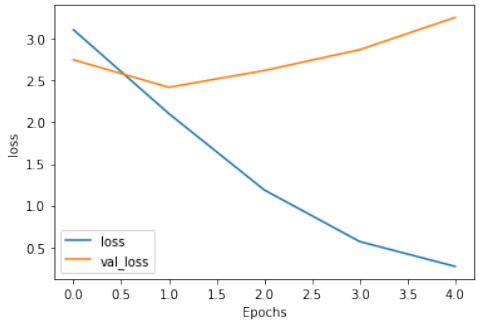
Figura 24: Arquitectura en Keras de los modelos en la sección D

falta modificar la salida de la capa *embedding*. Por lo tanto, no será necesario utilizar la capa *flatten* presente en las secciones previas. En este caso se ha utilizado una única capa oculta SimpleRNN. Estas capas corresponden a la implementación en Keras de las redes neuronales recurrentes en su versión más sencilla, explicadas con mayor detalle en la sección 2.4. Los tamaños elegidos para las capas de estos modelos han sido 64 y 128 neuronas, y los parámetros empleados han sido los mismos que en las secciones previas.

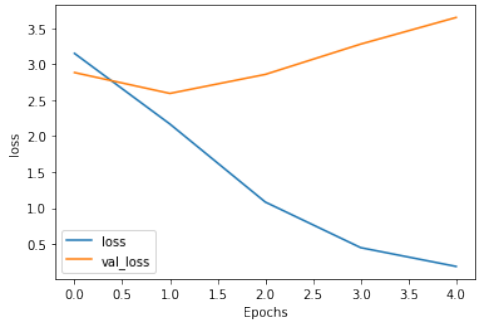
Modelo	Neuronas en capa SimpleRNN	Test Accuracy	Test Loss	Precision	Recall	F1-Score
D1	64	35.21 %	2.416	0.36	0.35	0.35
D2	128	29.42 %	2.602	0.31	0.29	0.289

Cuadro 5: Resultados de los modelos en la sección D.

El modelo de esta categoría que ha dado mejores resultados ha sido el D1, con una capa SimpleRNN de 64 neuronas. Aun así, los resultados son bastante mejorables si los comparamos con los obtenidos en las secciones anteriores.



(a) $D1$



(b) $D2$

Figura 25: Gráficas obtenidas de la función de pérdida en los modelos D1 y D2.

Si observamos las gráficas sobre la función de pérdida (figura 25), notamos que el valor de pérdida en la validación empieza a crecer en una etapa muy temprana del entrenamiento.

Esto nos indica que el uso de este tipo de capas no es efectivo para este problema, ya que la red empieza a perder capacidad de predicción (debido al sobreajuste) sin haber alcanzado una exactitud comparable con la de los modelos anteriores. En la siguiente sección, intentamos encontrar una mejora utilizando capas LSTM en lugar de las SimpleRNN.

6.3. Redes neuronales LSTM (Sección E)

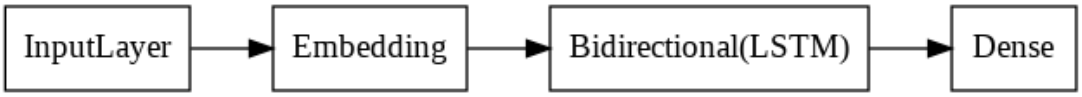


Figura 26: Arquitectura en Keras de los modelos en la sección E

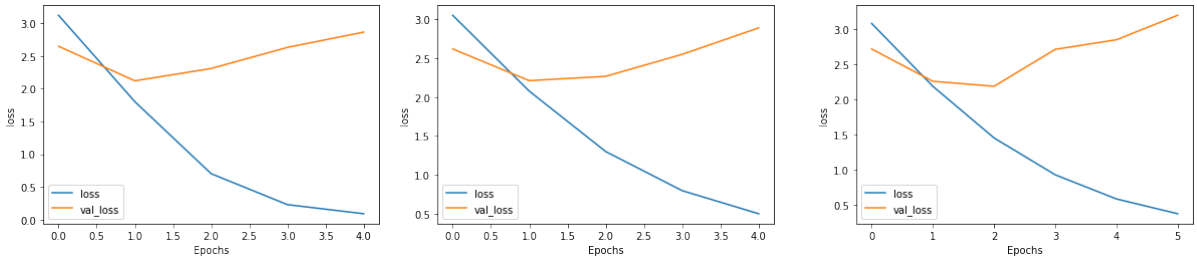
Como podemos ver, la única diferencia los modelos de esta sección y los de la sección D, es el uso de capas ocultas LSTM en lugar de SimpleRNN. Estos se han empleado debido a las diferencias mencionadas en el apartado 2.4.2, esperando obtener una mejora en los resultados. Los tamaños de capa empleados en la LSTM han sido 128, 256 y 512, y los demás parámetros vuelven a ser los mismos empleados en las secciones anteriores.

Modelo	Tamaño de capa LSTM	Test Accuracy	Test Loss	Precision	Recall	F1
E1	128	41.09 %	2.183	0.43	0.41	0.41
E2	256	41.01 %	2.224	0.44	0.41	0.41
E3	512	42.26 %	2.188	0.43	042	0.42

Cuadro 6: Resultados de los modelos en la sección E.

En estas pruebas, los modelos que mejores resultados han ofrecido han sido el E2 y el E3. De hecho, los resultados obtenidos son comparables con los de la sección A, dado que los valores obtenidos son muy similares.

Podemos notar que la función de pérdida en los modelos de esta sección tiene una trayectoria muy similar a la del modelo A2, de la sección A. Por lo tanto, conviene intentar mejorar los resultados utilizando más capas. Utilizaremos también el modelo E3 para comparar los modelos siguientes junto al A2, ya que es el que mejores resultados ha dado para esta sección.



(a) $A2$ (b) $E2$ (c) $E3$

Figura 27: Gráficas obtenidas de la función de pérdida en los modelos A2, E2 y E3.

6.3.1. Aumento de capas LSTM (Sección F)

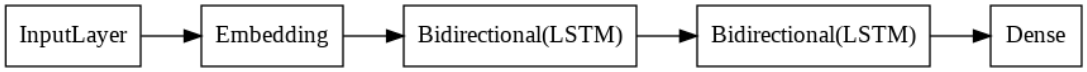


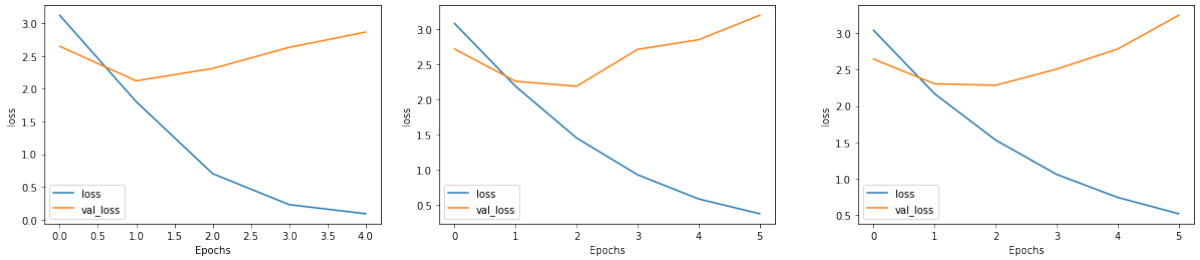
Figura 28: Arquitectura en Keras de los modelos en la sección F

Una vez estudiados los modelos con una única capa LSTM, intentamos mejorar los resultados añadiendo una segunda capa LSTM sin cambiar el resto de parámetros. Para la primera capa LSTM, se han utilizado tamaños de 128 y 256, mientras que para la segunda se han utilizado tamaños de 64, 128, 256 y 512. Los resultados obtenidos se encuentran en la tabla 7.

Modelo	Tamaño de capas		Test Accuracy	Test Loss	Precision	Recall	F1
F1	128	64	37.02 %	2.320	0.39	0.37	0.37
F2	128	128	37.14 %	2.296	0.39	0.37	0.37
F3	256	128	37.89 %	2.266	0.40	0.38	0.38
F4	256	256	39.22 %	2.322	0.41	0.39	0.39
F5	256	512	40.22 %	2.299	0.42	0.40	0.41

Cuadro 7: Resultados obtenidos para los modelos de la sección F.

Ninguno de los resultados obtenidos supera a los de los modelos anteriormente destacados (A2 y E3). El modelo con mejores resultados dentro de esta sección es el F5, aunque los resultados son mejorables. Sin embargo, podemos destacar que utilizar 256 como tamaño de la primera capa parece ofrecer mejores resultados que utilizar 128.



(a) $A2$ (b) $E3$ (c) $F5$

Figura 29: Gráficas obtenidas de la función de pérdida en los modelos A2, E3 y F5.

La gráfica de la función de pérdida (figura 29) nos muestra que el aprendizaje del modelo F5 es muy similar al de los mejores modelos estudiados hasta ahora. Dado que estos modelos utilizan un número y tipo de capas distinto, podemos suponer que la ausencia de mejora con las pruebas realizadas se debe a los datos de entrada. No obstante, en la siguiente prueba seguimos intentando mejorar los resultados intentando combinar capas LSTM y densas.

6.4. Modelos basados en la combinación de distintas capas (Secciones G y H)

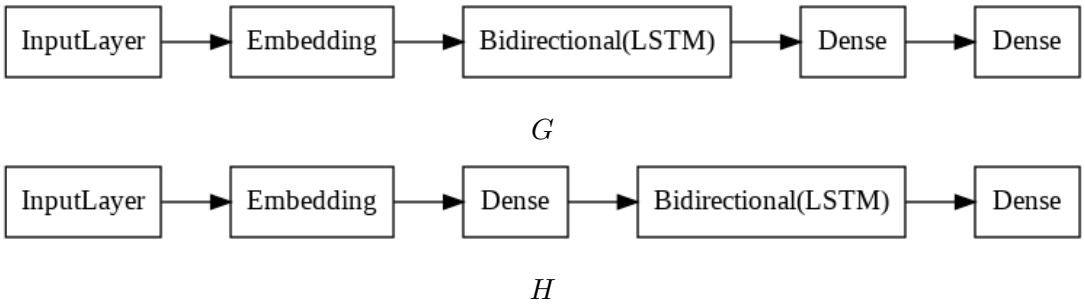


Figura 30: Arquitecturas de los modelos en las secciones G y H.

Ya que tanto el uso de una sola capa densa como el de una capa LSTM han dado los mejores resultados en las pruebas anteriores, intentamos encontrar modelos combinando ambas. Para ello, se han estudiado los modelos de las secciones G y H. Los modelos de estas dos secciones se diferencian en el orden que tienen los tipos de capas.

Para la sección G, se han estudiado modelos con una capa LSTM y una capa densa, en ese orden. En el caso de la sección H, se ha utilizado una capa densa y posteriormente una capa

LSTM. El resto de parámetros son los mismos que en las secciones anteriores, y los tamaños de capa empleados han sido 64, 128, 256 y 512 para la capa densa, mientras que para la LSTM el tamaño ha sido constante de 256.

Modelo	Tipo y tamaño de capa		Test	Test	Precision	Recall	F1
	Primera capa	Segunda Capa	Accuracy	Loss			
G1	LSTM (256)	Dense (64)	36.38 %	2.371	0.37	0.36	0.36
G2	LSTM (256)	Dense (128)	36.98 %	2.307	0.38	0.37	0.36
G3	LSTM (256)	Dense (256)	38.93 %	2.288	0.43	0.39	0.39
G4	LSTM (256)	Dense (512)	40.88 %	2.318	0.42	0.41	0.41
H1	Dense (64)	LSTM (256)	40.76 %	2.276	0.43	0.41	0.41
H2	Dense (128)	LSTM (256)	42.74 %	2.210	0.45	0.43	0.43
H3	Dense (256)	LSTM (256)	39.91 %	2.208	0.42	0.40	0.40
H4	Dense (512)	LSTM (256)	41.67 %	2.161	0.44	0.42	0.42

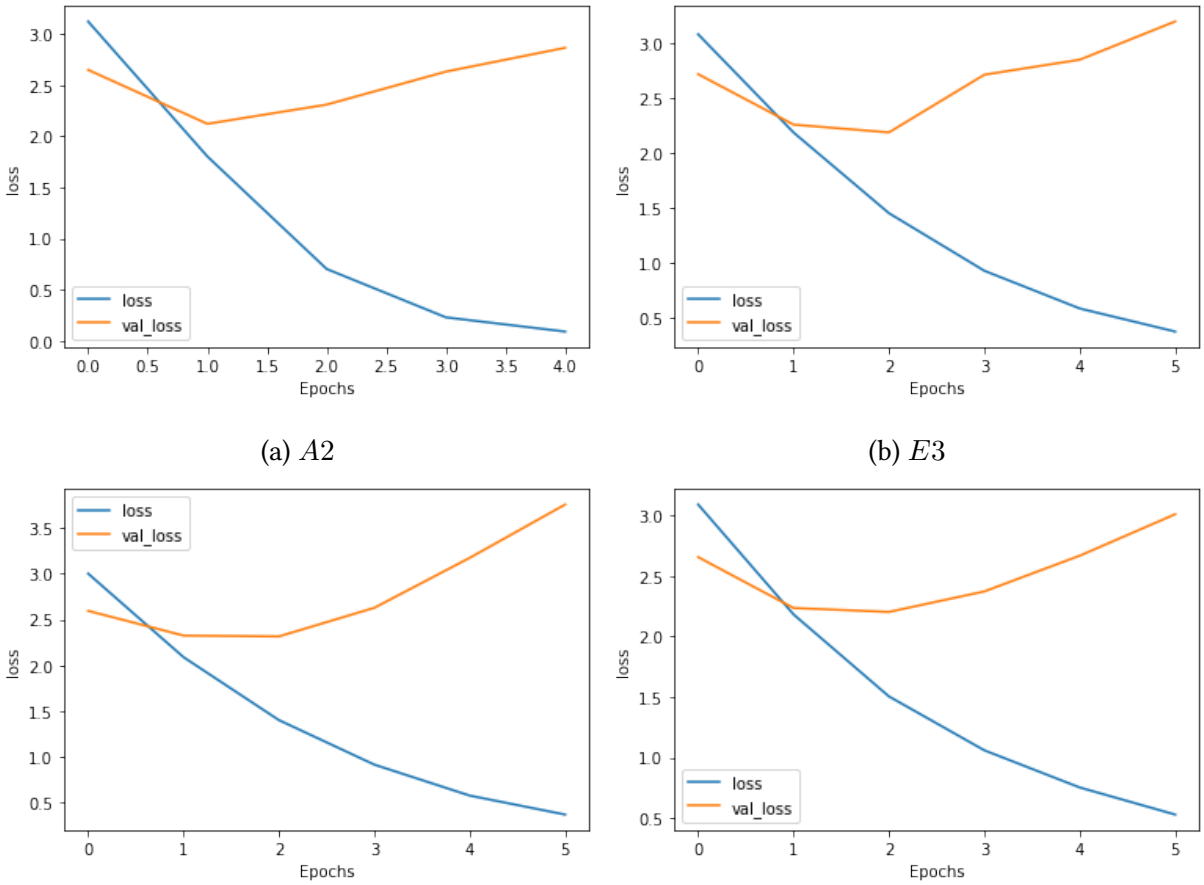
Cuadro 8: Resultados obtenidos en los modelos de las secciones G y H

De los resultados obtenidos de esta prueba podemos destacar los del modelo H2, que parecen ser similares a los de los mejores modelos A2 y E3, superándolo en el valor de exactitud para este último. Otro aspecto destacable de esta prueba es la mejora en los resultados al utilizar la capa densa en primer lugar respecto a utilizar la LSTM, ya que los modelos de la sección H dan mejores resultados que los de la sección G en todos los aspectos.

Algunos de los resultados obtenidos son comparables a los mejores modelos obtenidos previamente, pero no existe una mejora notable. Dado que estamos alcanzando unos valores máximos muy similares en las métricas de los mejores modelos, intentaremos cambiar la tasa de aprendizaje de estos para comprobar si esta limitación en la mejora se debe a este parámetro de entrenamiento.

6.5. Cambios en la tasa de aprendizaje (Sección I)

Dado que ninguna de las pruebas realizadas previamente ha conseguido que los modelos mejoren los resultados obtenidos, cabe estudiar la posibilidad de que se deba a problemas en el aprendizaje. Por ello, hemos intentado cambiar la tasa de aprendizaje tomando los modelos



(a) $A2$ (b) $E3$

(c) $G4$ (d) $H2$

Figura 31: Gráficas obtenidas de la función de pérdida para A2, E3, G4 y H2.

A2 y H2.

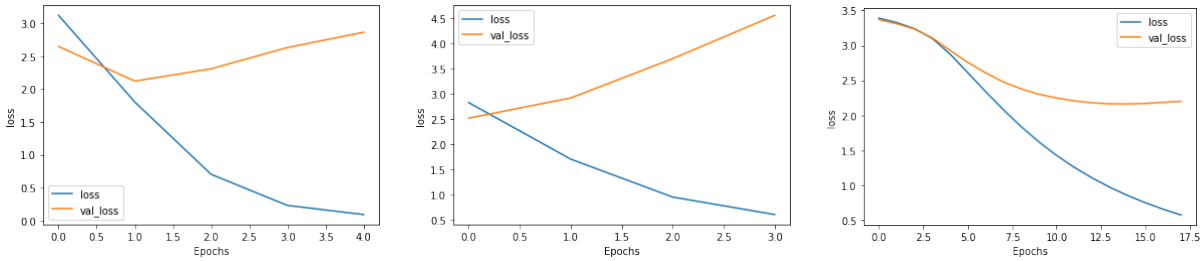
Los modelos I1 e I2 tienen la misma arquitectura y tamaño de capa que el A2 (figura 20) , y los modelos I3 e I4 tienen la misma arquitectura y tamaño de capa que el modelo H2 (figura 30). El único parámetro cambiado ha sido la tasa de aprendizaje, siendo el resto de parámetros los mismos que se han usado en el resto de secciones. Los modelos I1 e I3 utilizan una tasa de aprendizaje de 1^{-2}, mientras que los modelos I2 e I4 utilizan una tasa de 1^{-4}.

Los cambios realizados en la tasa de aprendizaje parecen empeorar los resultados obtenidos en los modelos A2 y H2, dando lugar a valores menores en todas las métricas empleadas. En el caso de los modelos I2 e I4, el impacto parece ser menor. Aun así, siguen siendo peores que los resultados de los modelos en los que se basan.

Las gráficas obtenidas de la función de pérdida (figura 32) nos muestran que cambiar la tasa de aprendizaje aleja el modelo del mínimo que buscamos. Por lo tanto, podemos concluir

Modelo	Learning rate	Tipo , tamaño y parámetros de capa		Test Accuracy	Test Loss	Precision	Recall	F1
		Primera capa	Segunda Capa					
I1	1e-2	Dense (256)		31.37 %	2.564	0.34	0.32	0.30
I2	1e-4	Dense (256)		41.51 %	2.218	0.43	0.42	0.42
I3	1e-2	Dense (128)	LSTM (256)	36.59 %	2.432	0.38	0.37	0.36
I4	1e-4	Dense (128)	LSTM (256)	38.47$	2.335	0.40	0.39	0.38

Cuadro 9: Modelos obtenidos cambiando la tasa de aprendizaje.



(a) $A2$ (b) $I1$ (c) $I2$

Figura 32: Gráficas obtenidas de la función de pérdida en los modelos A2, H1, y H2.

que la tasa de aprendizaje de los modelos previos (1^{-3}) es aceptable.

6.6. Métodos tradicionales

Método empleado	Test accuracy
Random forest	9.97 %
Gaussian Naive Bayes	4.22 %
Máquinas de soporte vectorial	5.26 %
Regresión logística	4.74 %

Cuadro 10: Resultados obtenidos mediante métodos tradicionales

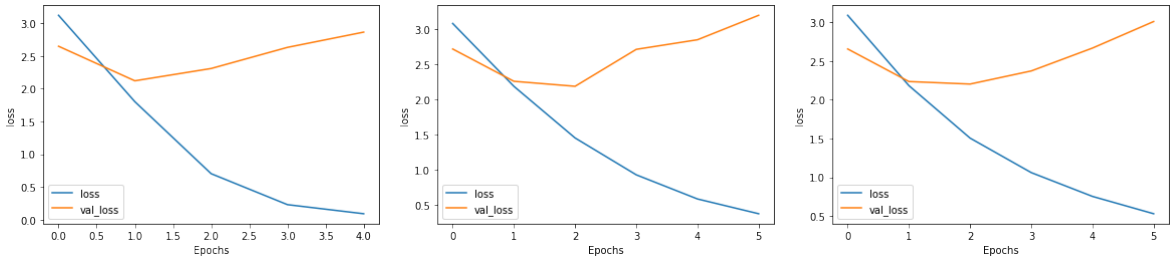
Los resultados obtenidos utilizando métodos tradicionales son considerablemente peores que los obtenidos utilizando redes neuronales. Teniendo en cuenta que estos métodos suelen ser muchos menos efectivos en este tipo de problemas que las técnicas basadas en redes neu-

ronales, era el resultado esperado. La exactitud del algoritmo *Random forest* es aproximada al 10 % en un problema con 30 clases, por lo que podemos destacar que las predicciones obtenidas están lejos de la obtención de valores aletorios (si fuera así tendríamos una exactitud arpoximada del 3,33 %)

7 Discusión

7.1. Discusión de los resultados

Teniendo en cuenta los resultados obtenidos, cabe destacar que los mejores modelos obtenido tienen resultados similares entre ellos, aunque utilicen distintos tipos de capa. Destacamos entonces como mejor modelo obtenido el **A2**, ya que obtiene los mejores resultados en todas las métricas empleadas (expuestas en el apartado 5.5).



(a) $A2$ (b) $E3$ (c) $H2$

Figura 33: Gráficas obtenidas de la función de pérdida para A2, E3 y H2.

Si comparamos las gráficas de la función de pérdida para los mejores modelos obtenidos (A2, E3 y H2) notamos muchas similitudes en el entrenamiento. Además, los resultados obtenidos no superan el 50 % para ningún modelo, lo cual indica claros problemas de calidad en el aprendizaje.

Además, a la vista la distribución de categorías del dataset (Figura 34), comprobamos que hay mucha diferencia entre la cantidad de datos disponibles para cada categoría. Es por ello que en el preprocesamiento de estos tuvimos que eliminar una gran cantidad de datos para equilibrarlos, con lo cual se ha perdido mucha información. Es bastante probable que esta sea una de las causas que han provocado limitaciones en el entrenamiento.

Una solución a este problema sería obtener más datos asociados a las categorías con pocos

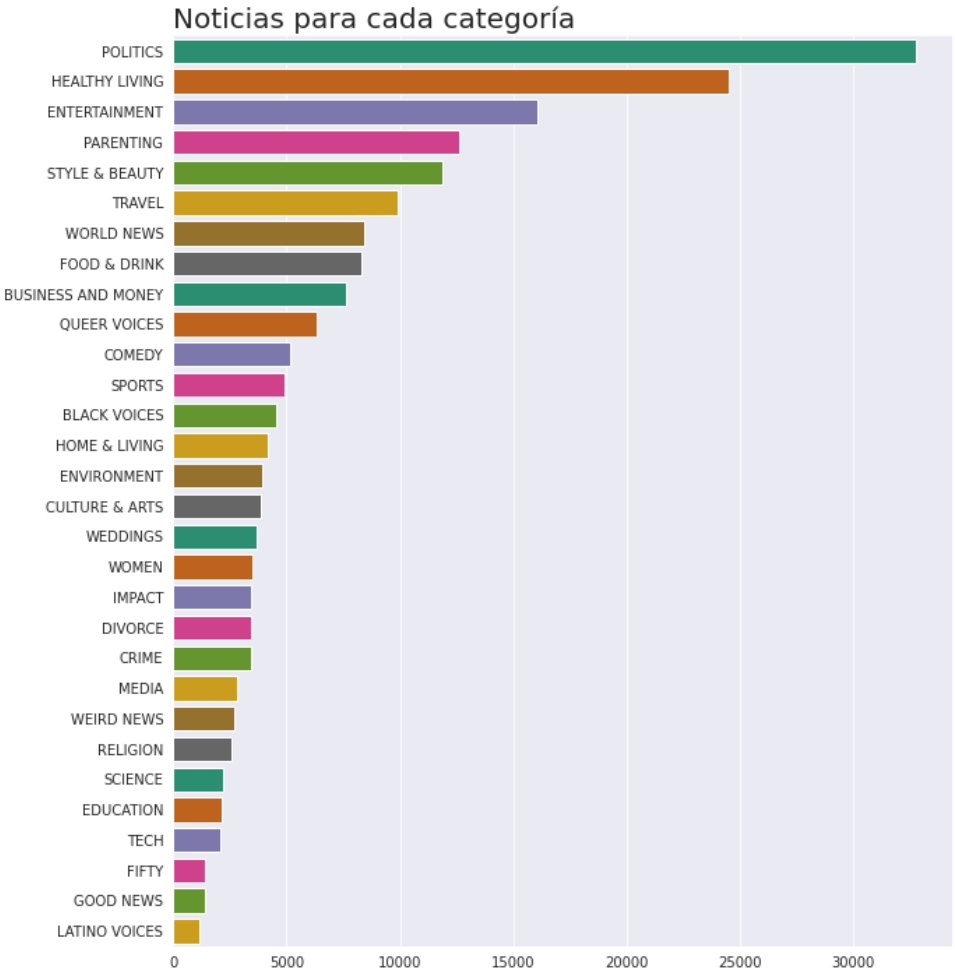


Figura 34: Distribución de categorías en el dataset.

datos. Así, al equilibrar el conjunto de datos perderíamos mucha menos información, de manera que el aprendizaje sería más largo dado que la red necesita más tiempo para alcanzar un mínimo absoluto en la gráfica de pérdida.

Aun así, podemos comprobar en la matriz de confusión del modelo A2 (Figura 35) que la red encuentra las relaciones entre titulares y categorías, dado que vemos que la diagonal de la matriz de confusión está claramente marcada y los errores obtenidos no están concentrados en ningún par de categorías. Ya que las matrices de los modelos E3 y H2 son muy similares a esta (figuras 36 y 37) podemos concluir que la red cumple con el propósito especificado, aunque con una exactitud bastante mejorable.

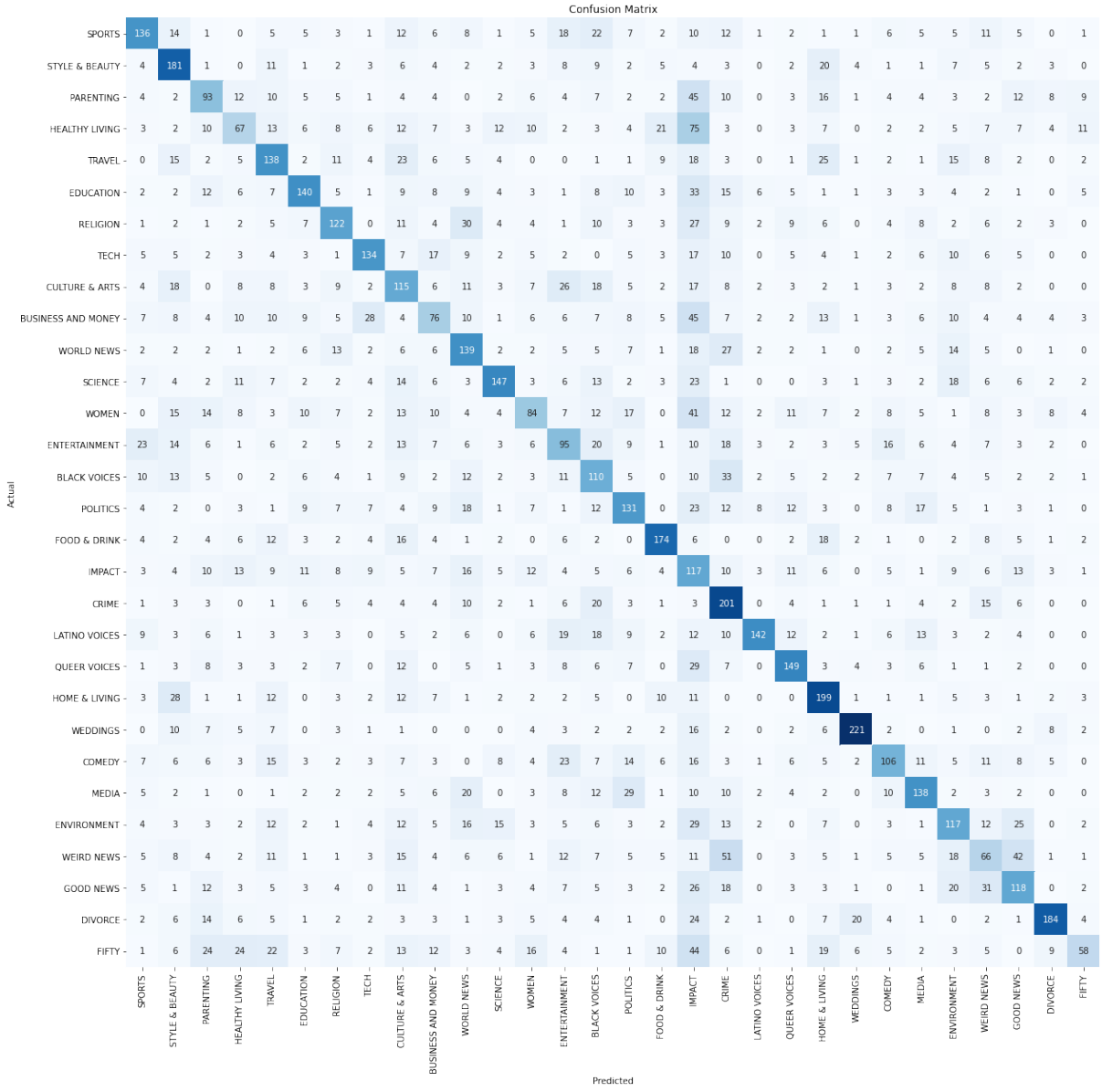


Figura 35: Matriz de confusión del modelo A2.

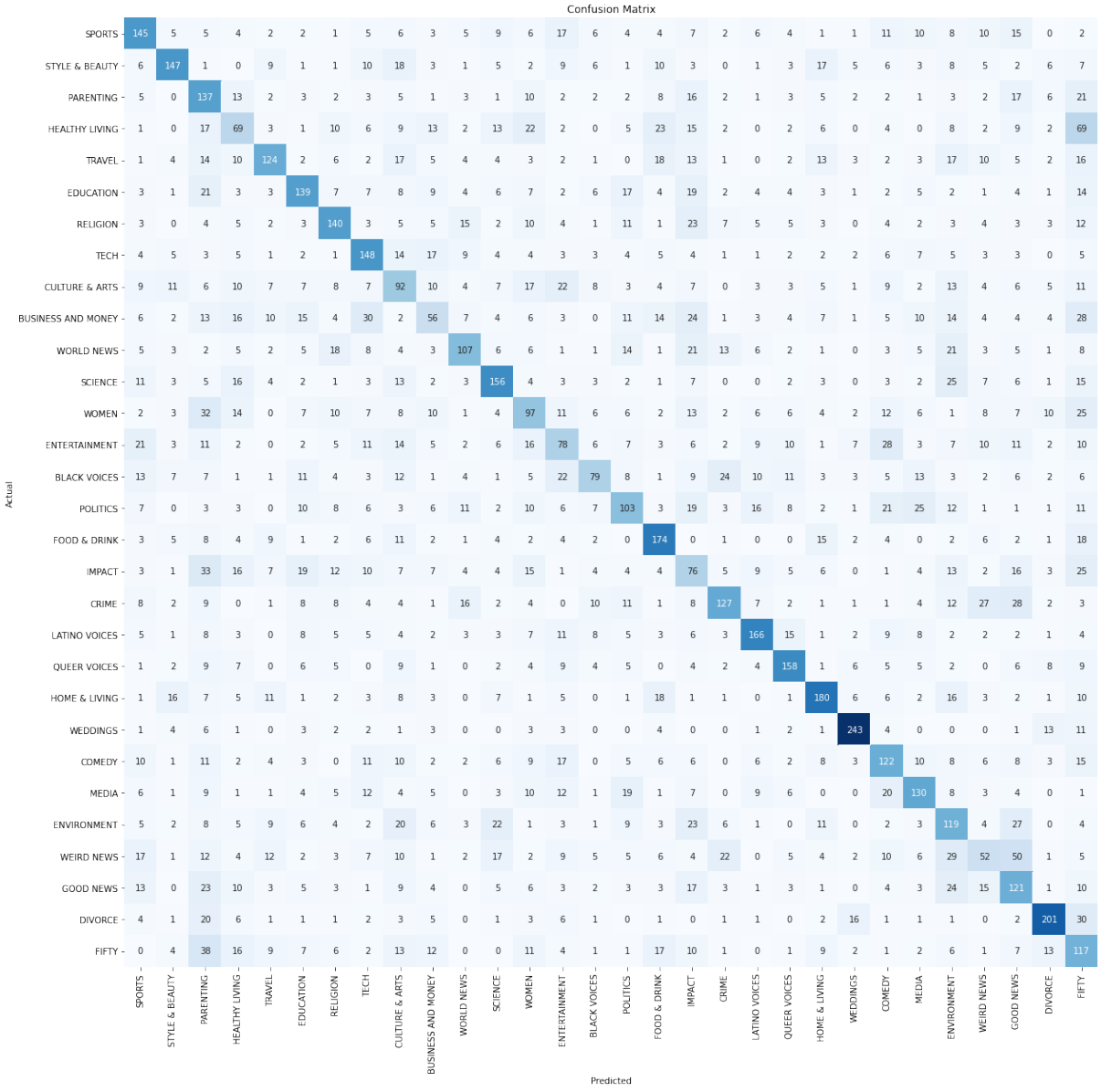


Figura 36: Matriz de confusión del modelo E3.

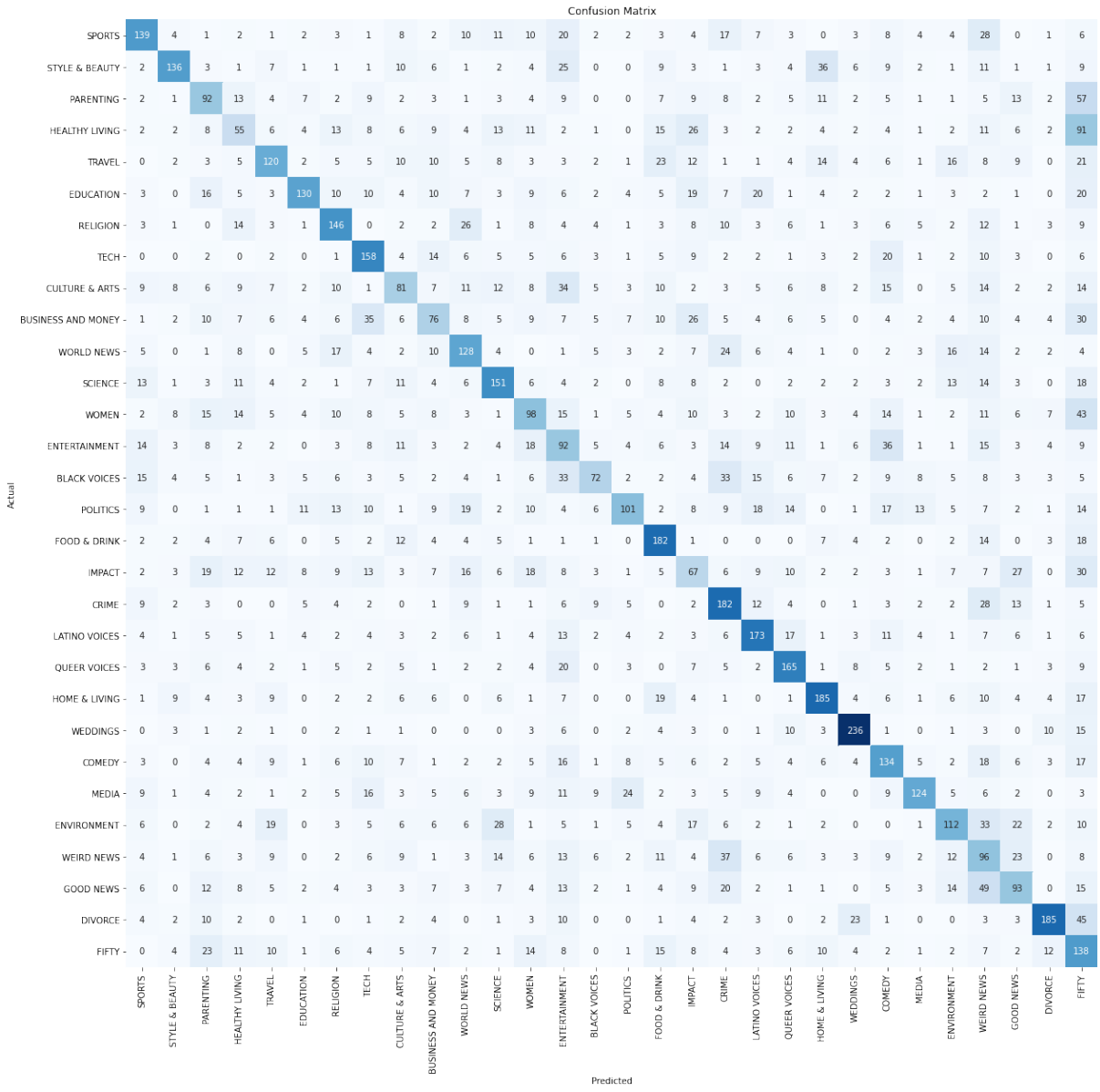


Figura 37: Matriz de confusión del modelo H2.

8

Conclusiones y Líneas Futuras

8.1. Conclusiones

En este trabajo se ha desarrollado un clasificador de categorías de noticias en función de su titular. Para ello, se han probado multitud de modelos de red neuronal con una amplia variedad de parámetros distintos. El mejor modelo conseguido ha sido, sorprendentemente, uno de los obtenidos en la primera prueba, utilizando un sola capa dense con 512 neuronas.

Dado que este modelo ha alcanzado únicamente una exactitud aproximada del 43 %, cabe destacar como posible trabajo futuro el uso de otros conjuntos de datos con categorías más específicas y un equilibrio mayor en la representación de cada categoría. Aun así, notamos que el aprendizaje realizado en este modelo es razonable dada la gran representación de la diagonal de la matriz de confusión, que indica una clara tendencia a dar una clasificación correcta.

Respecto al futuro del uso de este tipo de tecnologías en el ámbito periodístico, la posibilidad de ofrecer recomendaciones personalizadas al lector y poder encontrar maneras más eficientes de gestionar los artículos escritos resulta indispensable para cualquier medio de comunicación que pretenda adaptarse a las nuevas tecnologías. Además, este tipo de herramientas ya se están empleando de manera efectiva en redes sociales y otras plataformas digitales, así que es cuestión de tiempo que se adopten tanto en el sector periodístico como para otro tipo de medios digitales.

Referencias

[1] *OJDinteractiva MAYO 2021 - TOTALES - TRAFICO NACIONAL E INTERNACIONAL*, 2021. dirección: https://www.ojdinteractiva.es/medios-digitales?tipo_medio=offer.

[2] Statista, *Main source of information about news and current affairs in selected countries worldwide as of June 2020*, https://www.statista.com/statistics/198765/main-source-of-international-news-in-selected-countries/, (A 11/6/2021), 2020.

[3] A. Turing y J. Haugeland, *Computing machinery and intelligence*. MIT Press Cambridge, MA, 1950.

[4] B. Szabłowski, *What is an artificial neuron and why does it need an activation function?* Oct. de 2020. dirección: https://towardsdatascience.com/what-is-an-artificial-neuron-and-why-does-it-need-an-activation-function-5b4c1e971d80.

[5] V. Zhou, *Neural Networks From Scratch*. dirección: https://victorzhou.com/series/neural-networks-from-scratch/.

[6] P. S. -, By, -, P. S. a. P. Sharma, P. Sharma e I. a. P. Sharma, *Keras Dense Layer Explained for Beginners*, oct. de 2020. dirección: https://machinelearningknowledge.ai/keras-dense-layer-explained-for-beginners/.

[7] V. Stuart, *How do Neural Networks Remember¿*, ago. de 2018. dirección: https://persagen.com/2018/08/31/memory_in_rnn.html.

[8] *Understanding LSTM Networks*. dirección: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[9] S. Hochreiter y J. Schmidhuber, «Long short-term memory,» *Neural computation*, vol. 9, n.º 8, págs. 1735-1780, 1997.

[10] K. Team, *Keras documentation: LSTM layer*. dirección: https://keras.io/api/layers/recurrent_layers/lstm/.

[11] *Natural Language Processing Market Size, Share & Forecast 2025*. dirección: https://www.kbvresearch.com/natural-language-processing-market/.

[12] IPlens, *"This article was automatically written by Tencent Dreamwriter robot"*, mar. de 2021. dirección: https://iplens.org/2021/02/24/this-article-was-automatically-written-by-tencent-dreamwriter-robot/.

[13] R. Misra, *News Category Dataset*, jun. de 2018. DOI: 10.13140/RG.2.2.20331.18729.

[14] *Tokenization and Text Data Preparation with TensorFlow & Keras*. dirección: https://www.kdnuggets.com/2020/03/tensorflow-keras-tokenization-text-data-prep.html.

[15] *Word embeddings | Text | TensorFlow*. dirección: https://www.tensorflow.org/text/guide/word_embeddings.

[16] D. Ziganto, *Model Tuning (Part 2 - Validation & Cross-Validation)*, ene. de 2018. dirección: https://dziganto.github.io/cross-validation/data%20science/machine%20learning/model%20tuning/python/Model-Tuning-with-Validation-and-Cross-Validation/.

[17] R. Parmar, *Demystifying Optimizations for machine learning*, sep. de 2018. dirección: https://towardsdatascience.com/demystifying-optimizations-for-machine-learning-c6c6405d3eea.

[18] B. I. C. Education, *What is Overfitting?* Dirección: https://www.ibm.com/cloud/learn/overfitting.

[19] Hawlader, *Decision Tree Machine Learning: Some Real Life Example*, oct. de 2020. dirección: https://www.fossguru.com/decision-tree-machine-learning-some-real-life-example/.

[20] *Random Forest - Overview, Modeling Predictions, Advantages*, abr. de 2021. dirección: https://corporatefinanceinstitute.com/resources/knowledge/other/random-forest/.

[21] V. Maheshwari, *LOGISTIC REGRESSION*, ene. de 2019. dirección: https://medium.datadriveninvestor.com/logistic-regression-18afd48779ce.

[22] P. Majumder, *Gaussian Naive Bayes*, feb. de 2020. dirección: https://iq.opengenus.org/gaussian-naive-bayes/.

[23] *An Introduction to Support Vector Machines (SVM)*, jun. de 2017. dirección: https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/.

Apéndice A
Especificaciones del código

En este apéndice se muestra la estructura de la implementación de los modelos, así como el funcionamiento del código asociado. Las rutas empleadas corresponden a directorios en Drive, por lo que será necesario cambiarlas si se quiere volver a ejecutar.

Los datos empleados[13] están incluidos en el fichero *News_Category_Dataset_v2*, en formato *.json*. Para cargarlos se ha utilizado la función *read_json()* de la librería Pandas, devolviendo un objeto del tipo *DataFrame*. Para procesarlos, además de funciones propias de Python, hemos usado las funcion *apply()* del objeto cargado, para realizar cambios en los datos que contiene. también hemos usado la función *shuffle()* de la librería scikit-learn para reordenar los datos de manera aleatoria. La lista de adverbios eliminados ha sido obtenida de la librería nltk (*Natural Language Toolkit*).

En la construcción de los modelos se han empleado las clases implementadas en Keras asociadas con las capas que contienen, para crear el modelo instanciando la clase *Model* con las capas asociadas. Concretamente, en todos los modelos estudiados hemos usado instancias de la clase *Input* para la capa de entrada y la clase *Embedding* para los *word embeddings*. Además, la clase *Dense* ha sido instanciada para las capas densas (incluyendo la de salida), la clase *SimpleRNN* para las capas de las redes neuronales recurrentes simples (sección D) y la clase *LSTM* en las capas de las redes LSTM. La clase *Flatten* se ha utilizado como capa para cambiar las dimensiones de los datos.

Para definir otros parámetros del modelo se han creado instancias de la clase *Adam* en el caso el optimizador y la clase *CategoricalCrossentropy* como función de pérdida. Estos parámetros se han introducido en el modelo utilizando la función *compile()*, que configura el modelo para permitir entrenarlo, usando los parámetros como argumentos. Tras el entrenamiento, los modelos obtenidos se han guardado en el mismo directorio, con los mismos nombres que tienen en la tabla B.

Una vez construido y configurado el modelo, hemos empleado el método *fit()* para entrenarlo, utilizando como argumentos los parámetros de entrenamiento definidos en la sección 5.3, además de definir límites y condiciones en el tiempo de entrenamiento. Estos límites se basan en un número de iteraciones (*epochs*) que el modelo realiza sobre los datos (en nuestro caso definido a 100) y una instancia de la clase *EarlyStopping*, en la cual indicamos que se detenga el entrenamiento si el valor de la función de pérdida decrece durante tres iteraciones seguidas.

El método *fit()*, además de entrenar la red, nos devuelve el historial de entrenamiento, que guardamos utilizando la librería Pickle para poder estudiarlos. En cambio, para guardar los modelos y los pesos de sus neuronas, hemos usado la función *save* del modelo. Todos los modelos e historiales de entrenamiento vienen incluidos en el fichero, junto al código.

Las matrices de confusión expuestas en este trabajo han sido tomadas utilizando la función *confusion_matrix()* de la librería scikit-learn y los datos de la tabla mediante las funciones *evaluate()* en la clase *Model* y la función *classification_report()* de scikit-learn. Las imágenes de las arquitecturas para los modelos se han obtenido utilizando la función *plot_model* de la clase *Model*. Tanto las gráficas empleadas para la función de pérdida como los diagramas expuestos en la sección 4 han sido tomados utilizando la librerías Matplotlib y Seaborn.

Apéndice B
Listado de modelos y resultados

Cuadro 11: Categorías y frecuencias

Modelo	Learning rate	Tipo , tamaño y parámetros de capa			Test Accuracy	Test Loss	Precision	Recall	F1
		Primera capa	Segunda Capa	Tercera capa					
A1	1e-3	Dense (128)			42.14 %	2.142	0.47	0.42	0.43
A2	1e-3	Dense (256)			43.31 %	2.112	0.46	0.43	0.44
A3	1e-3	Dense (512)			42.90 %	2.139	0.46	0.43	0.44
A4	1e-3	Dense (1024)			43.14 %	2.168	0.45	0.43	0.43
B1	1e-3	Dense (128)	Dense (128)		37.64 %	2.344	0.40	0.38	0.38
B2	1e-3	Dense (256)	Dense (256)		37.18 %	2.332	0.37	0.37	0.36
B3	1e-3	Dense (512)	Dense (512)		37.48 %	2.347	0.40	0.38	0.37
B4	1e-3	Dense (1024)	Dense (1024)		38.53 %	2.330	0.39	0.39	0.38
C1	1e-3	Dense (256)	Dense (256)	Dense (256)	26.86 %	2.629	0.31	0.27	0.25
C2	1e-3	Dense (512)	Dense (512)	Dense (512)	24.72 %	2.676	0.26	0.25	0.23
C3	1e-3	Dense (1024)	Dense (1024)	Dense (1024)	24.07 %	2.708	0.27	0.24	0.22
D1	1e-3	SimpleRNN (64)			35.21 %	2.416	0.36	0.35	0.35
D2	1e-3	SimpleRNN (128)			29.42 %	2.602	0.31	0.29	0.289
E1	1e-3	LSTM (128)			41.09 %	2.183	0.43	0.41	0.41
E2	1e-3	LSTM (256)			41.01 %	2.224	0.44	0.41	0.41
E3	1e-3	LSTM (512)			42.26 %	2.188	0.43	042	0.42
F1	1e-3	LSTM (128)	LSTM (64)		37.02 %	2.320	0.39	0.37	0.37

Cuadro 11: Categorías y frecuencias

Modelo	Learning rate	Tipo , tamaño y parámetros de capa			Test Accuracy	Test Loss	Precision	Recall	F1
		Primera capa	Segunda Capa	Tercera capa					
F2	1e-3	LSTM (128)	LSTM (128)		37.14 %	2.296	0.39	0.37	0.37
F3	1e-3	LSTM (256)	LSTM (128)		37.89 %	2.266	0.40	0.38	0.38
F4	1e-3	LSTM (256)	LSTM (256)		39.22 %	2.322	0.41	0.39	0.39
F5	1e-3	LSTM (256)	LSTM (512)		40.22 %	2.299	0.42	0.40	0.41
G1	1e-3	LSTM (256)	Dense (64)		36.38 %	2.371	0.37	0.36	0.36
G2	1e-3	LSTM (256)	Dense (128)		36.98 %	2.307	0.38	0.37	0.36
G3	1e-3	LSTM (256)	Dense (256)		38.93 %	2.288	0.43	0.39	0.39
G4	1e-3	LSTM (256)	Dense (512)		40.88 %	2.318	0.42	0.41	0.41
H1	1e-3	Dense (64)	LSTM (256)		40.76 %	2.276	0.43	0.41	0.41
H2	1e-3	Dense (128)	LSTM (256)		42.74 %	2.210	0.45	0.43	0.43
H3	1e-3	Dense (256)	LSTM (256)		39.91 %	2.208	0.42	0.40	0.40
H4	1e-3	Dense (512)	LSTM (256)		41.67 %	2.161	0.44	0.42	0.42
I1	1e-2	Dense (256)			31.37 %	2.564	0.34	0.32	0.30
I2	1e-4	Dense (256)			41.51 %	2.218	0.43	0.42	0.42
I3	1e-2	Dense (128)	LSTM (256)		36.59 %	2.432	0.38	0.37	0.36
I4	1e-4	Dense (128)	LSTM (256)		38.47$	2.335	0.40	0.39	0.38

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA